NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

EXPLORATION OF FIBRE CHANNEL AS AN AVIONICS INTERCONNECT FOR THE 21ST CENTURY MILITARY AIRCRAFT

by

Shawn P. Hendricks

September 2000

Thesis Advisor: Russell W. Duren
Second Reader: John C. McEachen

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE		Form Approved OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 2000	Engineer's Thesis

4. TITLE AND SUBTITLE: Title (Mix case letters)
Exploration of Fibre Channel as an Avionics Interconnect for the 21st Century Military Aircraft

5. FUNDING NUMBERS

6. AUTHOR(S) Shawn P. Hendricks

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
N/A

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

Avionics architectures are evolving from "Federated" systems consisting of highly specialized black boxes connected together via MIL-STD-1553 and ARINC 429 data buses to "Integrated" and "Distributed" architectures. These new architectures contain high data-rate sensors, parallel processors, and shared memory with high levels of integration. These systems require a new interconnection system that overcomes the limitations of older standards. One such interconnection system is Fibre Channel. This thesis evaluates Fibre Channel as avionics interconnection standard. It begins by defining the requirements and measures of performance for an interconnection system suitable for the new avionics architectures. The requirements address technical performance, affordability, reliability, sustainability, and maintainability considerations. The Fibre Channel standards are then compared to the requirements for the avionics interconnection system. In order to perform a technical performance evaluation of a switched fabric avionics interconnection system, a computer simulation model was developed. The OPNET Modeler® tool from OPNET, Inc. was used to model the components of an advanced avionics system. The results of this simulation demonstrated that Fibre Channel meets all the performance requirements of an avionics interconnect.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Fibre Channel, interconnect, avionics, bandwidth, modeling, simulation	196
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

EXPLORATION OF FIBRE CHANNEL AS AN AVIONICS INTERCONNECT FOR THE 21ST CENTURY MILITARY AIRCRAFT

Shawn P. Hendricks
Lieutenant Commander, United States Navy
B.S., United States Naval Academy, 1989

Submitted in partial fulfillment of the
requirements for the degrees of

MASTER OF SCIENCE IN AERONAUTICAL ENGINEERING
AND
AERONAUTICAL AND ASTRONAUTICAL ENGINEER

from the

NAVAL POSTGRADUATE SCHOOL
September 2000

ABSTRACT

Avionics architectures are evolving from "Federated" systems consisting of highly specialized black boxes connected together via MIL-STD-1553 and ARINC 429 data buses to "Integrated" and "Distributed" architectures. These new architectures contain high data-rate sensors, parallel processors, and shared memory with high levels of integration. These systems require a new interconnection system that overcomes the limitations of older standards. One such interconnection system is Fibre Channel. This thesis evaluates Fibre Channel as avionics interconnection standard. It begins by defining the requirements and measures of performance for an interconnection system suitable for the new avionics architectures. The requirements address technical performance, affordability, reliability, sustainability, and maintainability considerations. The Fibre Channel standards are then compared to the requirements for the avionics interconnection system. In order to perform a technical performance evaluation of a switched fabric avionics interconnection system, a computer simulation model was developed. The OPNET Modeler® tool from OPNET, Inc. was used to model the components of an advanced avionics system. The results of this simulation demonstrated that Fibre Channel meets all the performance requirements of an avionics interconnect.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I. INTRODUCTION 1
A. BACKGROUND 1
B. REQUIREMENTS 2
1. Technical Requirements 3
a. Real Time and Deterministic 3
b. Throughput 4
c. Protocol 5
d. Flow Control 5
2. Fiscal Requirements 6
a. Scalability, Supportability and Maintainability 6
b. Commercial Off the Shelf 7
c. Open Systems 8
C. SUMMARY 9

II. FIBRE CHANNEL AS A SOLUTION 11
A. FIBRE CHANNEL BACKGROUND 11
B. FIBRE CHANNEL DESIGN 12
1. Serial Interface 12
2. High Throughput 13
3. Scalability 13
a. Point-to-Point Links 14
b. Arbitrated Loop 14
c. Switched Fabric 16
4. Deterministic and Real Time 16
a. Point-to-Point and Arbitrated Loop 17
b. Switched Fabric 17
5. Flow Control 18
a. Buffer-To-Buffer Flow Control 18
b. End-To-End Flow Control 19
6. Reliable Transmission/Error Detection and Correction 19
a. 8b/10b Coding 19
b. Cyclic Redundancy Check 20
7. Multi-Protocol 20
8. Commercial Off the Shelf 20
9. American National Standards Institute Standard 21
a. FC-PH/FC-FS 21
b. FC-AE 22
c. FC-SL 22
C. CONCLUSION 22

III. MODEL CONSTRUCTION 25
A. OPNET MODELER 25
1. Network Editor 25

2. Node Editor 26
3. Process Editor 26
4. Link Editor 27
5. Packet Editor 27
6. PDF Editor 27
B. OPNET MODELER IMPLEMENTATION OF FIBRE CHANNEL 28
1. The Node 28
a. Packet Generator Object 29
b. The Processor Module 30
c. The Transmitter Module 30
d. The Receiver Module 30
2. The Switch 31
a. The Processor Module 32
b. The Transmitter/Receiver Modules 32
3. The Link Model 32
4. The Packet Model 33
a. The Fibre Channel Frame 33
b. The R_RDY Primitive 35
IV. FINITE STATE MACHINE DESCRIPTIONS 37
A. OVERVIEW 37
B. THE NODE FINITE STATE MACHINE 37
1. Initialization Path 38
a. Init State 38
b. Objid? State 38
c. Subnet? State 39
d. Function? State 39
e. I Talk To State 40
f. Computers State 41
g. Others State 41
2. Traffic Generation Path 41
a. Idle State 42
b. Get_pkt State 42
c. B_to_b State 42
d. B_plate State 43
e. CL1_xmt State 44
f. CL2_xmt State 44
g. CL3_xmt State 44
h. E_to_e State 44
3. Receive and Process Path 45
a. Rcv State 45
b. R_rdy_rcv State 45
c. Snd_rrdy State 46
d. FC_0 State 46
e. FC_1 State 47
f. Check Credit State 48

4. Transmit Path 48
a. *Topo_build State* 48
b. *Route State* 49
c. *Xmt_pkt State* 50
C. THE SWITCH FINITE STATE MACHINE 50
1. Initialization Path 51
a. *Init State* 51
b. *Objid? State* 52
c. *Subnet? State* 52
d. *Con_out State* 52
e. *Set_link_cost State* 53
2. Process Path 53
a. *Idle State* 53
b. *Type State* 53
c. *R_rdy State* 54
d. *Frame State* 54
e. *Dec_buf State* 54
f. *F_bsy State* 54
3. Transmit Path 55
a. *Add_route State* 55
b. *Sndrrdy State* 55
c. *Xmt State* 55
V. MODEL VALIDATION 57
A. OVERVIEW 57
B. TESTS 57
1. Throughput 57
a. *Theory* 57
b. *Model* 58
c. *Test and Test Conditions* 58
d. *Results* 59
2. Overhead 59
a. *Theory* 60
b. *Model* 60
c. *Test and Test Conditions* 60
d. *Class 3 Traffic Results* 61
d. *Class 2 Traffic Results* 61
3. End-to-End Delay 62
a. *Theory* 62
b. *Long Distance Model* 64
c. *Test and Test Conditions* 64
d. *Short Distance Model* 67
e. *Test and Test Conditions* 67
f. *Results* 67
4. Load Balancing 68
a. *Theory* 68

b. Load Balance Model 69
c. Test and Test Conditions 69
d. Results 70
e. Conclusion 73
C. SUMMARY 73
VI. DESIGN, CONSTRUCTION AND TEST OF POSSIBLE AIRCRAFT ARCHITECTURES 75
A. INTRODUCTION 75
B. DESIGN OF A SIMPLE SYTEM 75
1. System Parameters 76
2. Throughput Calculations 77
3. Buffer-to-Buffer Credit Calculations 78
a. Node Buffer-to-Buffer Calculations 78
b. Switch Buffer-to-Buffer Credit Calculations 79
4. End-to-End Credit Calculations 80
5. System Test 82
a. Buffer-to-Buffer Credit Check 82
b. Throughput 83
c. End-to-End Delays 85
VII. CONCLUSION 87
A. WHAT THE SIMULATION SHOWED 87
1. Determinism and Latency 87
2. Problems With Displays 88
3. Burden of Administrative Overhead 88
4. Difficulty In Analytical Estimates For Complex Systems 89
B. WHY FIBRE CHANNEL CAN WORK 89
C. WHY FIBRE CHANNEL MAY NOT WORK 90
D. FOLLOW ON WORK 91
1. Session and Exchange Layers 91
2. Probability Density Function Models of Real Systems 91
3. Comparison of Simulation Data to Real System 91
APPENDIX A. COMPARISONS OF OTHER INTERCONNECT SYSTEMS 93
APPENDIX B. OPNET NODE FINITE STATE MACHINE CODE 95
APPENDIX C. OPNET SWITCH FINITE STATE MACHINE CODE 145
APPENDIX D. MATLAB CODE USED DURING SIMULATION 167
LIST OF REFERENCES 173
INITIAL DISTRIBUTION LIST 175

LIST OF FIGURES

Figure 1. Evolution of Avionics Architectures. From Ref. [1]. 1
Figure 2. Fibre Channel Topologies. From Ref. [9] 13
Figure 3. Point-to-Point Configuration. From Ref. [10]. 14
Figure 4. Arbitrated Loop Ring Network. From Ref. [10]..... 15
Figure 5. Arbitrated Loop Ring Network With Central Hub. From Ref. [10]. 15
Figure 6. Switched Fabric topology. From Ref. [10]. 16
Figure 7. OPNET Implementation of a Fibre Channel Node..... 29
Figure 8. OPNET Model of the Fibre Channel Switch. 32
Figure 9. OPNET Representation of a Fibre Channel Frame. 33
Figure 10. OPNET Representation of a Fibre Channel Frame. 36
Figure 11. Fibre Channel Node Finite State Machine..... 37
Figure 12. Fibre Channel Switch Finite State Machine. 51
Figure 13. Model Used to Verify Throughput and Overhead. 58
Figure 14. Results of the Simple Throughput Validation Test. 59
Figure 15. Results of the Class 3 Overhead Verification Test. 61
Figure 16. Results of the Class 2 Overhead Verification Test. 62
Figure 17. Long Distance Model..... 64
Figure 18. Results of the Class 3 End-to-End Verification Test. 65
Figure 19. Results of the Class 2 End-to-End Verification Test. 66
Figure 20. Short Distance End-to-End Test Verification Model 67
Figure 21. Results of the Short Distance End-to-End Delay Verification Test. 68
Figure 22. Load Balance Verification Model..... 69
Figure 23. Load Balance Verification Test Result for DEL_COST = 0.5 71
Figure 24. Load Balance Verification Test Result for DEL_COST = 0.10 sec 71
Figure 25. Relationship Between Settling Time and DEL_COST..... 73
Figure 26. Simple Aircraft Avionics System 76
Figure 27. A Histogram Showing the Distribution of End-to-End Delays 80
Figure 28. Buffer-to-Buffer Credit Resets For the Full System Test..... 83
Figure 29. Throughput From All Nodes, 50/50 Mix of Class 2 and 3 Traffic 84
Figure 30. End-to-End Delays for the Example System 86

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

Table 1. Data Rate and Throughput Projections. From Ref. [1]. 4
Table 2. Digital Network Requirements. From Ref. [1]. 9
Table 3. Parallel vs. Serial. After Ref. [4]..... 12
Table 4. Frame Size Breakdown 58
Table 5. Data From the Load Balance Verification Tests..... 72
Table 6. Desired Values for Node Throughput 76
Table 7. R_RDY Delay = 100.0nsec, Switch Processing Delay = 500.0nsec 77
Table 8. Calculated Frame Interarrival Arguments..... 78
Table 9. Time for A Node to receive R_RDY 79
Table 10. End-to-End Credit Calculations 81
Table 11. Increase in Throughput Due to Fibre Channel Protocol 84
Table 12. End-to-End Statistics..... 86
Table 13. Digital Network Requirements. After Ref. [1]..... 90
Table B-1. List of State Variables Used in the Node Finite State Machine..... 95
Table B-2. List of Temporary Variables Used in the Node Finite State Machine..... 96
Table C-1. List of State Variables Used in the Switch Finite State Machine 145
Table C-2. List of Temporary Variables Used in the Switch Finite State Machine 145

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

The author would like to acknowledge the support of the Joint Strike Fighter Program Office. Specifically, CAPT David Wooten, CDR Scott Orren and Mr. Patrick Moore were especially helpful during the research phase of this thesis.

While the entire staff at OPNET was excellent, the author would like to acknowledge the following persons for their patience and extraordinary help during the software development: Alain Cohen, Yev Gurevich, Pradeep Singh, and the entire Santa Clara OPNET office especially Farzam Fallah, and Dylan Morris.

Lastly, the author would like to thank Dr. Russ Duren and Dr. John McEachen for their support, insight and timely feedback of technical issues.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. BACKGROUND

Since the 1970's avionics have rapidly progressed from primarily analog to primarily digital. A paradigm shift has occurred in which aircraft architectures have migrated from many highly specialized "Black boxes" loosely interconnected, to a system of general-purpose computers sharing many specialized tasks and vast amounts of data. Figure 1 shows the evolution of the different architectures.

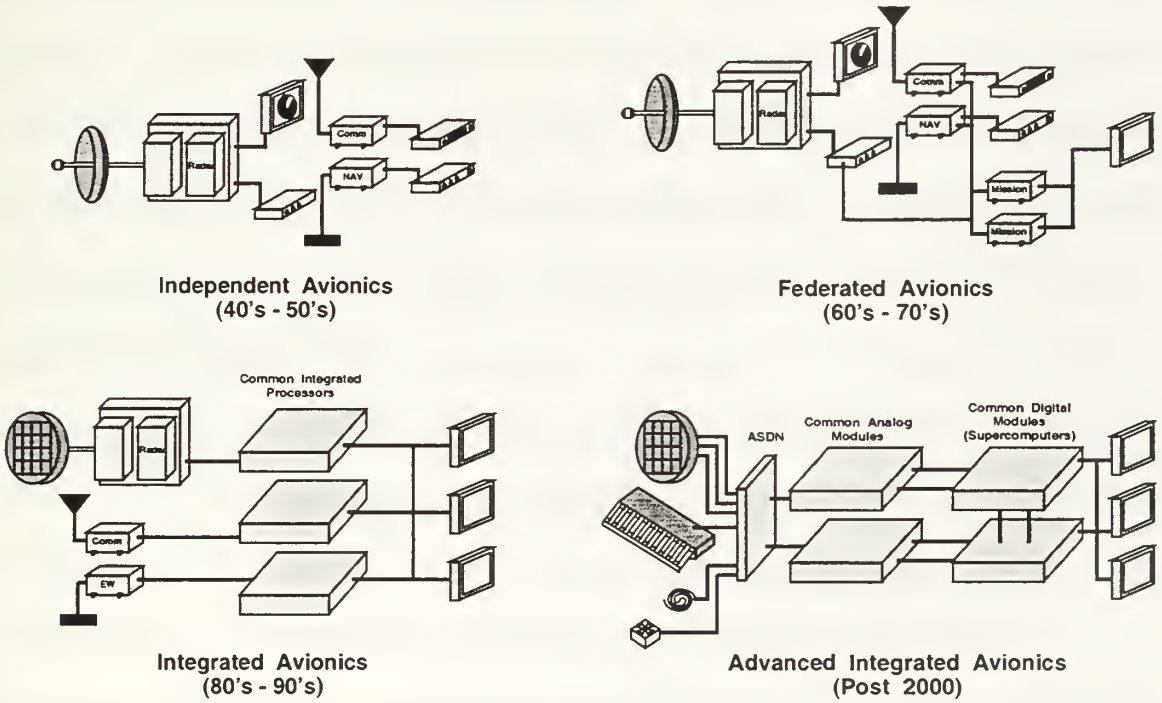

Figure 1. Evolution of Avionics Architectures. From Ref. [1].

During this same period of time computational speeds have increased from 1 Hz, to rates today that exceed 1GHz and show no signs of plateau. In fact, Moore's Law states that computer speeds will double every eighteen months into the foreseeable future

[Ref. 2]. Today's military aircraft also have requirements to carry sensors that can receive, display and relay video in real time. With all of these advances and shifts in architecture philosophy one vital technology has lagged, the bandwidth of the interconnection architecture.

In the early 1970's the Federated Architecture was born, and the MIL-STD-1553 data bus was introduced to military aircraft. The 1553 bus had a nominal throughput of 1.0 megabyte per second (MBs) [Ref. 3]. Thirty years later 1553 is still used in most military aircraft even though processing speeds have increased by six orders of magnitude. While the military has lagged the civilian sector with regards to introducing computers with cutting edge processing speed, it can be shown that the civilian sector is just as reticent when it comes to system bandwidth. In 1964 the throughput of the "state of the art" IBM System/360 was 1.25 MBs, over thirty years later this had grown to only 4.5MBs an increase of only four times. Amdahl's law of I/O bandwidth required states "for every CPU instruction per second one bit of I/O bandwidth is required [Ref. 4]." Clearly, this shows that there is a need for better I/O interconnects.

B. REQUIREMENTS

During the process of selecting a new avionics interconnect, the requirements for this interconnect must be defined. These requirements will involve both technical requirements and fiscal requirements.

1. Technical Requirements

The following paragraphs will outline the major technical requirements of any avionics system. While this list should not be considered complete, it will address most of the concerns of any system designer/program manager.

a. Real Time and Deterministic

Military aircraft are constrained by some requirements that commercial applications are not. First, they must have low latency. In the shared memory systems of the integrated avionics architectures of the 21st Century, processors will require low latency so that they may operate at peak efficiency without waiting for data. The Joint Advanced Strike Technology Program (JAST) Avionics Architecture Definition states that "high latency interconnects often result in very low efficiency parallel processors. Recent experiments in commercial message passing parallel processor systems have demonstrated that even high latency interconnects can have a devastating effect on processor efficiency [Ref. 1]."

In the Military, computer systems are by their very nature real time systems. High priority message traffic must be delivered on time even in the presence of low priority traffic. The Oxford Dictionary of Computing [Ref. 5] defines a real time system as follows: "A real time system is any system in which the time at which output is produced is significant. This is usually because the input corresponds to some movement in the physical world, and the output has to relate to that same movement." A second requirement is a high level of determinism. The system must behave in a predictable manner. Dr. Phil Dowe of The University of Tasmania in Australia [Ref. 6] defines determinism as "one who's future and past states can be predicted from a complete

knowledge of the current state and the laws of nature." The system must be able to deliver error-free information in a well-defined time frame.

A military aircraft must be designed to operate in a combat environment. It is to be expected that at some time the aircraft will sustain damage. Since the avionics are such an integral part of the aircraft, involved in such things as flight controls, weapons systems and navigation, the data cannot be interrupted and alternate data paths must be available in the event that one or more "wires" is damaged. Also, if one of the processors on the bus begins to fail there must be a mechanism to detect and correct or ignore errors. All of this must occur in real time and in a deterministic fashion with as little impact of mission capability as possible.

b. Throughput

As the aircraft has an increased need for more information from high bandwidth systems, the throughput available over the bus must also increase. This high throughput is also required to minimize the latency of data passing between processors. In 1994 the JAST mission systems Integrated Product Team (IPT) came up with a series of throughput requirements for the avionics bus. These requirements are summarized in Table 1.

Table 1. Data Rate and Throughput Projections. From Ref. [1].

Application (Year 2010)	**Data Rate Projection (per channel)**	**Throughput Projection (includes preprocessing)**
IRST	120 - 200 Mbits/sec	4 - 10 GOPS
FLIR	120 - 160 Mbits/sec	3 - 10 GOPS
ADAS		
SIT Awareness	150 - 700 Mbits/sec	4 - 10 GOPS
Navigation	150 - 700 Mbits/sec	1 - 2 GOPS
Threat Warning	150 - 700 Mbits/sec	1 - 4 GOPS

RGHPRF	280 Mbits/sec	
ASLC + RGHPRF	280 Mbits/sec	2-15 GOPS
SAR	200-800 Mbits	
EW-RF (RWR/ESM)	1 -2 Gbits/sec	0.5 - 2.0 GOPS
EW-EO (Missile Warning)	SEE ADAS ABOVE	SEE ADAS ABOVE
EW-C3 (Special Receiver)	200 - 400 Mbits/sec	0.5 - 1.0 GOPS
EW-EO (Laser Warning)	50 - 100 Mbits/sec	50 - 100 MIPS
Total EO		15-25 GOPS
Total Radar		2-15 GOPS
Total EW suite		5 - 11 GOPS
Total CNI suite (WBDL+GPS+IFF)	TBD	30 - 50 GOPS*

* Normally done by specialized preprocessors

Clearly, based on this data, the required throughput of the interconnect will need to be at least 2 Gbits/sec. However, given the rapid increases in processor technology, and the need for the aircraft to be viable for thirty or more years, a more realistic goal should be 4 to 10 Gbits/sec.

c. Protocol

Different computers "speak" different languages. They are addressed by various protocols. One device might use the Small Computer Scalable Interface (SCSI) while another might be communicating in Internet Protocol (IP) while yet another could be streaming video over Asynchronous Transfer Mode (ATM). The interconnect must be able to handle all of these (and many others) with a minimum amount of overhead.

d. Flow Control

Memory in a contained system is a finite commodity. For each message that is sent there must be adequate memory resources available in the receiver to process the message. If this is not the case it is very possible that messages could get lost

unbeknownst to the sender. This would cause confusion between the nodes and probably crash the system.

Flow control is ensuring that the receiver has the required resources available to receive the message or the sender is notified somehow that resources are not available and some buffering at the sender must take place. The next generation system must have some form of flow control.

2. Fiscal Requirements

The military procurement system is constrained by the budget process. All procurement decisions must be made with the goal of getting the best product that meets the design requirements for the least amount of money. Avionics systems are not exempt from this requirement. The JAST Mission Systems Integrated Product Team's Avionics Architecture Definition [Ref. 1] is quoted below.

> Affordability is of primary importance to the JAST. Therefore the JAST avionics architecture must be predominantly driven by cost considerations. The avionics architecture should seek to reduce life cycle costs, especially development costs. Affordability constraints require the architecture to support an open system concept, insertion and use of commercial and openly available military technology/standards, and the reuse of software.

Avionics now account for up to 30% of the total aircraft cost [Ref. 1]. This figure will continue to increase in the foreseeable future, and now can be viewed as an area where "smart" procurement decisions will produce large savings.

a. Scalability, Supportability and Maintainability

Another change in aircraft and their procurement has been the length of time that aircraft are expected to remain in service. Currently, the U.S. Military is flying aircraft that are expected to have a useful service life of thirty plus years. This means that military aircraft configurations are continually being changed and upgraded. Through the

use of transformer coupling, the MIL-STD-1553 bus is very tolerant to the addition or subtraction of nodes. Any replacement to MIL-STD-1553 would also have to support the easy addition, removal and maintenance of any nodes on the bus.

Not only will the aircraft be upgraded over time, but also a military aircraft in combat is continually changing its configuration throughout a combat sortie. The interconnection may have to interface with weapons that are themselves shared processors. High rates of data may be passed between the aircraft's internal avionics and the external stores. These stores will go away, and the system must be able to make a seamless transition from one configuration to another.

In order to facilitate ease of maintenance, as well as simplicity when upgrading, a bus with a minimum of wires would be desired. With fewer connections the probability of failure due to poor connections is reduced. Also the overall weight of the aircraft is reduced, allowing more of the weight budget to be allocated to weapons and fuel. Fewer adjacent wires also reduce the problems of unintentional electromagnetic interference. Finally, it would be good to eliminate one of the major timing problems associated with parallel busses, clock skew. Bus clock skew is where the bits become displaced from one another in time [Ref. 4]. This displacement can force the receiving node to reassemble and possibly reorder the data bits prior to processing, resulting in delays and bottlenecks. All of these factors imply that a serial bus would be the best candidate for the next generation aircraft.

b. Commercial Off the Shelf

Currently there are two philosophies that the government has adapted in order to maximize savings. The first is through the use of Commercial Off the Shelf

technology (COTS). The use of COTS greatly reduces the government's development cost, as well as providing a path for future upgrades. Use of COTS also requires some use of caution. There may be many hidden costs associated with COTS. Prior to procurement, careful analysis must be completed to ensure that the COTS part meets the needs of the government. These requirements include but are not limited to environmental constraints, interoperability between systems and maintainability.

c. Open Systems

The second philosophy used is the Open Systems Approach. The Naval Sea Systems Command (NAVSEA) Acquisition Support [Ref. 7] office defines an open system as follows:

> An Open System is a system that implements sufficient open (vendor independence, non-proprietary, publicly available, and widely accepted) specifications and standards for interfaces, services, and supporting formats to enable properly engineered components to be utilized across a wide range of systems with minimal changes, to interoperate with other components on local and remote systems, and to interact with users in a style that facilitates portability. An Open System is characterized by:
>
> - Well defined, widely used, non-proprietary interfaces/protocols.
> - The use of standards which are developed/adopted by industry recognized standards bodies.
> - All aspects of system interfaces defined to facilitate new or additional systems capabilities for a wide range of applications.
> - Explicit provision for expansion or upgrading through the incorporation of additional or higher performance elements with minimal impact on the system.

Conversely, NAVSEA defines closed systems in the following paragraph:

> Closed systems are regarded to be proprietary, secret, or patented, while open systems are based on standards which are agreed to and published by an accredited, consensus-based group. USD (A&T)'s memorandum of 29 November 1994 states that "open system specifications and standards are consensus-based public or nonproprietary specifications and standards for

systems and interfaces of hardware, software, tools, and architecture. Acquisition and upgrade costs will be minimized through the use of open systems architecture.

C. SUMMARY

Most of the technical requirements (as defined by the JAST Program Office) of the next generation avionics interconnection are summarized in the following table.

Table 2. Digital Network Requirements. From Ref. [1].

Requirement	Third Generation	Fourth Generation
Network Size	<32 connects	<256 connects
Data Rate-per path	400 Mbits/s	2.5 Gbits/s
Data Rate-aggregate	10+ Gbits/s	50-100 Gbits/s
Data Integrity-streaming	10E-7 BER	10E-7 BER
Data Integrity-packets	10E-10 BER	10E-10 BER
Packet size	Unlimited	Unlimited
Path Latency (μsec)	<100	<10
Link Control	Self-Routing	Self-Routing
Blocking	Limited	Non-Blocking
Packaging	SEM-E	SEM-E

Apart from the technical requirements some decisions are made based on fiscal concerns. First, the interconnect should be a commercially viable solution. It should be widely implemented and a fairly mature technology. This will allow the use of COTS, which will in turn increase the number of vendors available and drive down the component price. Second, an open systems approach should be used. This means that a well-defined, non-proprietary standard should be at the heart of the system.

THIS PAGE INTENTIONALLY LEFT BLANK

II. FIBRE CHANNEL AS A SOLUTION

A. FIBRE CHANNEL BACKGROUND

Currently there are several candidates for the interconnect technology of the 21st Century aircraft. This section will address how well Fibre Channel meets each of the requirements laid out in section one. Section three will compare Fibre Channel with some of the other interconnect technologies.

Computer processors are very fast. Application software is composed of many Mbytes of data. Unfortunately, no matter how fast the processor, the slowest process is a major factor in the overall time it takes the computer to complete a task. Typically that process is data access. The current computer interconnect technologies cannot supply data fast enough to the processor. As was noted earlier, the bandwidth of interconnect technology has only increased by factors of less than ten while the processors are operating at speeds that have increased by several orders of magnitude. All this leads to bottlenecks in the computer system. Fibre Channel was designed to address this shortfall [Ref. 4]. The major problem to be solved was a way to get large amounts of data from various remote "warehouses" of data storage elements (hard drives, tape or other media) to a processor running an application at a different site. Kembel [Ref. 4] states that, " . . . what was needed was a physical interface that was independent of any specific command set behavior and flexible enough to support different command sets and their associated architectures." In 1988 a study was begun on developing a standard that would address these concerns.

B. FIBRE CHANNEL DESIGN

The following paragraphs will discuss the relevant design points of Fibre Channel which may or may not make it suitable for use in an avionics environment.

1. Serial Interface

"Serial done right is faster and cheaper than parallel outside the chip." (Ed Lee, Fibre Channel Group, Ref. 8) Fibre Channel is a serial interface. Serial busses have many advantages over their parallel counter parts. Summarized in Table 3 are some of the tradeoffs between serial and parallel interfaces.

Table 3. Parallel vs. Serial. After Ref. [4].

Factor	Advantage Parallel	Advantage Serial
Wider data bus	Each clock pulse clocks in more bits	Less complex cables and connectors. Less weight, less cost.
		No clock skew
		Less I/O pins on integrated circuits.
		More benefit from logic density improvements.
		Lower probability of failure due to decreased number of parts and connections.
		No cross talk on adjacent wires.
Faster clock rate	More clock pulses per unit time equals more bits per unit time.	No clock skew. (As clock speed inc. clock skew becomes more of a factor.)
		No cross talk (Faster clock pulses have more cross talk and noise.)
		Does not decrease transmit/receive distance.
Multi-drop shared bus	Lower cost	Bandwidth is not shared.
		No added protocol overhead (arbitration etc.)
		Number of devices is not a factor of loading.

Up until recently, the advantage that made up for the increased complexity of the parallel bus was that for the same data rate the serial bus had to operate N times faster, where N is the bus width. Today's advances in semiconductor technology have allowed cost effective serial transmissions of over one gigabit per second [Ref. 4].

2. High Throughput

The current Fibre Channel standard, FC-PH full speed, is capable of operating at throughputs of one gigabit per second. Double-speed (200 Mbytes/sec) and quad-speed (400 Mbytes/sec) are being demonstrated today. [Ref. 4] Future developments in technologies such as Dense Wave Division Multiplexing (DWDM) could increase speeds to greater than 40 gigabits per second. Throughputs such as these are more than adequate to address the needs of future avionics systems, and show that Fibre Channel meets the throughput criteria of the 21st century aircraft.

3. Scalability

Fibre Channel is a full duplex bus that uses two cables between nodes of the system. One cable is the transmit fiber the other the receive fiber. Each set of two fibers and their associated connectors are refereed to as a link [Ref. 4]. Fibre Channel is designed to support three types of interconnections or links between subsystems, point-to-point, arbitrated loop and fabric. Examples of these are shown in Figure 2.

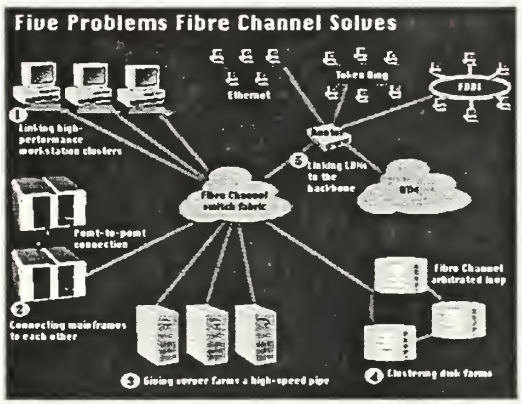

Figure 2. Fibre Channel Topologies. From Ref. [9]

a. Point-to-Point Links

The first and simplest interconnection is the point-to-point link shown in Figure 3.

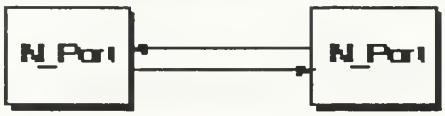

Figure 3. Point-to-Point Configuration. From Ref. [10].

The point-to-point only supports connection of two systems together. Scalability is impossible. A system cannot be added unless one of the two "old" systems is replaced with the "new" system. Also, there is no sharing of resources other than between the two connected systems. Clearly the point-to-point link is not a suitable interconnection scheme for the advanced integrated avionics architecture envisioned for the next generation aircraft.

b. Arbitrated Loop

Another way to connect Fibre Channel nodes together is by using an arbitrated loop (FC-AL). FC-AL was designed as a relatively inexpensive way for small companies to convert their remote storage to Fibre Channel. Two of the primary FC-AL topologies are the arbitrated loop ring and the arbitrated loop with hub.

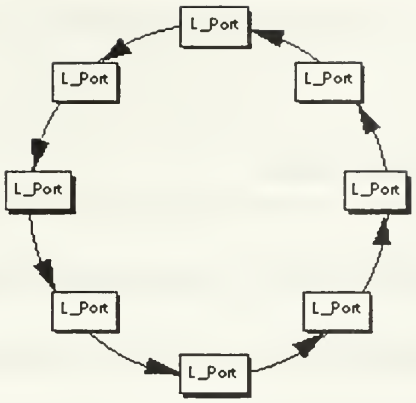

Figure 4. Arbitrated Loop Ring Network. From Ref. [10].

In this configuration messages are passed from sub-system to subsystem until the message or data gets to the appropriate end user.

Figure 5 shows the arbitrated loop with a central hub.

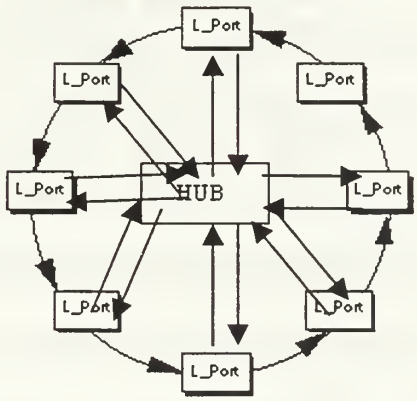

Figure 5. Arbitrated Loop Ring Network With Central Hub. From Ref. [10].

The hub in the center can be configured to allow for communications to continue even if one subsystem on the loop is incapacitated for some reason. Both configurations are scalable, and can have up to 127 systems on one loop. Since bandwidth is shared a more reasonable figure is 20 to 30 devices. It is however not truly "hot-swappable." Any time a device is added or removed from the topology the network must re-initialize

[Ref. 8]. Ed Lee of the Fibre Channel Group is quoted, "When compared to FC Switched Fabric, its sole benefit is its lower purchase price." [Ref. 8]

c. Switched Fabric

Fibre Channel's Switched Fabric (FC-SW) is a routing structure that examines the destination address in the header of the message and delivers the message to the intended recipient [Ref 4.]. There is no single path from one device to another, rather many possible paths of which one is chosen. Figure 6 shows one possible fabric configuration.

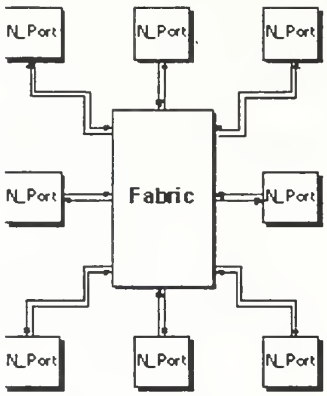

Figure 6. Switched Fabric topology. From Ref. [10].

The fabric is the most flexible of all the topologies. First the number of possible nodes is greater that 16 million, giving unparalleled scalability. (Compare this to the 31 possible devices on a MIL-STD-1553B bus [Ref. 2].) Also, unlike FC-AL, true live-insertion, plug and play is possible [Ref. 8]. Clearly, in terms of scalability, for avionics applications, the switched fabric is the best configuration.

4. Deterministic and Real Time

"The best way to ensure determinism and real time performance is through excess bandwidth." Ed Lee, The Fibre Channel Group.

a. Point-to-Point and Arbitrated Loop

The point-to-point architecture can be both deterministic and real time. Dedicated bandwidth ensures that the connection is only dedicated to one task, so that task is sure to be completed. Unfortunately, there is not much need for a two-system architecture, so it has no real practical use in an avionics environment.

It has been shown that FC-AL is moderately scalable. FC-AL has a reasonable address space, and from a strictly scalability standpoint could be used in small architecture applications. The main drawbacks for using FC-AL are determinism and real time performance. By its design, every device on the loop shares the bandwidth. The more devices that are in the architecture, the less excess bandwidth that exists, and hence the less likely that the task will get completed within a certain timeframe. Also, since there is only one way for messages to travel around the ring, additional protocols must be added to arbitrate who gets priority on the loop. This adds overhead and delays. Some members of the Fibre Channel industry have referred to FC-AL as the "Arbitrated Noose" [Ref. 8] because of these limitations. Lastly, each time a node is added or removed a Loop Initialization Protocol (LIP) is started. This LIP causes additional unpredictable delays further reducing determinism.

b. Switched Fabric

Switched fabric can be configured so that multiple, dedicated "conversations" can be conducted between devices. Due to the many paths available, very little arbitration is required. Also, as a truly "hot-swappable" configuration, no additional protocol is executed when the configuration of the system is changed. This allows much more deterministic behavior when compared to FC-AL. Finally, in fabric the bandwidth between any two devices does not have to be shared, giving switched

fabric a marked advantage over FC-AL in terms of aggregate bandwidth. This superior deterministic behavior makes switched fabric the logical topology choice if Fibre Channel is the network backbone.

5. Flow Control

It takes finite time for a receiver to process incoming traffic. If a second frame arrives while that processing is taking place it must be placed into some sort of buffer awaiting processing. Since there is only a finite amount of memory (hence a finite amount of buffers), the receiver must be able to regulate the rate at which frames are sent from the sender. If this was not done, frames could be sent to a node/switch without sufficient resources and that frame would be lost with no record of its demise. Flow control allows the receiver to control the rate at which the sender may send frames. All of the Fibre Channel topologies offer two types of flow control: buffer-to-buffer and end-to-end.

Both types of Fibre Channel flow control use a term called "Credit" to ensure that frames are only sent to nodes that have buffers to receive them. Just as there are two type of flow control there are two types of credit: buffer-to-buffer and end-to-end credit. Credit is granted during the login process and is based on the resources available in the nodes.

a. Buffer-To-Buffer Flow Control

Ensuring buffer-to-buffer flow control is mandatory for all frames. Buffer-to-buffer flow control simply ensures that the next node in the path has resources available to process the frame. For example, upon login a node has credit granted to it from the switch to which it is attached. The credit then decrements by one each time that the node then sends a frame. The node is free to send frames as quickly as it can up to

the point where this credit is exhausted. Credit is returned to the node through use of the Fibre Channel primitive R_RDY, which is explained in Chapter III. Buffer-to-buffer credit in no way assures that the final destination has the resources to process the frame. End-to-end flow control is responsible for that task.

b. End-To-End Flow Control

End-to-end flow control is a way to guarantee delivery of packets in Class 1, Class 2 and Class 6 messages. Much like buffer-to-buffer flow control, during the login process the receiving nodes grant end-to-end credit to the nodes that send them traffic. This means that end-to-end credit is maintained by a node for all other nodes with which it may communicate. Unlike buffer-to-buffer, having end-to-end credit with a receiving node ensures that ultimate receiver has resources to process the frame.

Much like buffer-to-buffer flow control a mechanism exists to replenish the end-to-end credit. It is the ACK frame. Unlike the R_RDY, the ACK is a full frame. It is considered a Frame Type 0 (FT-0) otherwise known as a Link Control Frame. Link Control Frames are described in more detail in Chapter III.

6. Reliable Transmission/Error Detection and Correction

All the error detection and correction features mentioned below are available in any of the Fibre Channel topologies.

a. 8b/10b Coding

Unlike parallel interfaces, serial interfaces cannot send a clock. The clock signal is required to define the time frame in which individual bits are valid. Fibre Channel uses a coding scheme known as 8b/10b coding to reconstruct the clock signal at the receive end [Ref. 4]. In 8b/10b ten bits are coded for each byte of data. 8b/10b coding has several benefits that make it a desirable way to send information between

systems. First it ensures a balanced transmission. A balanced transmission is defined as a transmission that for an arbitrary byte boundary an equal number of ones and zeros are broadcast, and whose frequency is approximately that of the transmission clock. This balanced transmission ensures the elimination of DC and low frequencies that in turn ensures the elimination of electrical and thermal transients [Ref. 8]. Another benefit is that the circuitry required to reconstruct the transmit clock is greatly simplified, and hence much cheaper. Lastly, this data protocol is independent of the media used (copper, optical etc.), and distance traveled [Ref. 8].

b. Cyclic Redundancy Check

A Cyclic Redundancy Check (CRC) is performed on each message transmitted through the system. The CRC uses the following 32-bit polynomial [Ref. 4]:

$$X^{32} + X^{26} + X^{23} + X^{22} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^5 + X^4 + X^2 + X + 1$$

The bit error rate (BER) for this polynomial is less than 10^{-12} [Ref. 4]. This equates to less than one error for each 15 minutes of operation at one Gbps [Ref. 8]. This BER should be more than sufficient for any avionics application.

7. Multi-Protocol

Fibre Channel is a data passing protocol. It is concerned only with the correct passing of data not the format of that data. This means that any protocol can be supported by a Fibre Channel interconnection scheme.

8. Commercial Off the Shelf

Fibre Channel was designed by the commercial industry. Research indicates that while Fibre Channel is being widely adapted for the Storage Area Network (SAN) industry, very few current military or rugged industrial applications could be found. Several vendors of Fibre Channel technology were contacted at the Fibre Channel

Technology Conference (FTCT 99), and none of those had any plans to enter into the military market. The lack of a clear COTS market such as exists for technologies like MIL-STD-1553, RS-232 or ARINC 429 busses make future availability of COTS Fibre Channel hardware with military applications somewhat less likely.

9. American National Standards Institute Standard

The American National Standards Institute (ANSI) supports all Fibre Channel standards and projects. The following paragraph was taken from the ANSI web page [Ref. 11]

> The American National Standards Institute (ANSI) has served in its capacity as administrator and coordinator of the United States private sector voluntary standardization system for 80 years. Founded in 1918 by five engineering societies and three government agencies, the Institute remains a private, nonprofit membership organization supported by a diverse constituency of private and public sector organizations.

The technical group that ANSI charters to oversee the development of these standards is the T11 group [Ref.12]. All current standards and projects are listed at the T11 web site [Ref. 12].

ANSI is a widely recognized standard and therefore Fibre Channel is suitable for the open systems approach for acquisition of systems for government use. The standards (and their current stage of development) that are most suitable for military application are described in the paragraphs below.

a. FC-PH/FC-FS

FC-PH is the base document for Fibre Channel. It contains the description of the data format as well as the physical characteristics of Fibre Channel. The latest approved version of FC-PH is FC-PH-3.

In 1998 a decision was made to consolidate the relevant sections (framing and signaling) from FC-PH, FC-PH-2, and FC-PH-3 and associated errata, annexes and amendments into a single document [Ref. 12]. For this reason future documents will probably refer to the key document as FC-FS as opposed to FC-PH.

b. FC-AE

FC-AE is a proposed standard that would allow Fibre Channel to transport information between avionics subsystems. It is currently in development, and a draft of the standard is listed as Ref. 13. Since this standard is only in draft form, there is some risk that it will not develop as forecast.

c. FC-SL

FC-SL stands for the slotted loop configuration of Fibre Channel. It is in development for use in very specific aerospace applications. FC-SL is being designed to provide real time data using a loop topology. It is similar in function to MIL-STD-1553B. There are three primary differences between FC-SL and FC-AL. First, a master element is designated that is responsible for creating time slots for the other ports on the loop. Second, there is no arbitration. The master device controls all access to the loop. Finally, slotted loop port is more complex and costly than an arbitrated loop port. [Ref. 4]

C. CONCLUSION

The background research on Fibre Channel yielded some promising results. It was clearly designed for scalability as well as high-bandwidth, low-latency, real-time communications. The development process is being conducted by a national standards organization in an open manner. These aspects lead to the conclusion that Fibre Channel could be used in an avionics architecture.

Once a qualitative assessment was made as to the suitability of Fibre Channel, a search was made for a tool to help design and evaluate avionics architectures using Fibre Channel as the interconnection protocol. Since the architecture of the 21st Century is supposed to utilize shared processing resources, it was determined that a tool that was able to conduct complex simulations of networks would be a good way to evaluate Fibre Channel. One such software package is OPNET Modeler, by OPNET Inc.

The next three sections describe OPNET Modeler in detail, the adaptation of OPNET to Fibre Channel, and lastly how one would use this tool in evaluation of an avionics system.

THIS PAGE INTENTIONALLY LEFT BLANK

III. MODEL CONSTRUCTION

A. OPNET MODELER

OPNET Modeler was first demonstrated at MIT in 1987. It was designed to be predictive network management software, which can boost R&D productivity, improve product quality, and reduce time-to-market. OPNET Modeler contains many models of existing hardware, allowing quick construction, test and analysis of many types of networks and protocols including, but not limited to ATM, Ethernet, Fast Ethernet, HTTP, and TCP-IP. For this project the strength of OPNET Modeler was in the ability to quickly model any type of packet behavior at many levels of abstraction. This was possible by use of the three primary editors included in the OPNET Modeler package. These are the Network Editor, the Node Editor and the Process Editor. Three other editors were used to a lesser extent: the Packet Editor, the Link Editor, and the PDF Editor.

1. Network Editor

The Network Editor is a graphical representation of a communications network. Networks are composed of Node and Link objects (developed in the Node and Link Editors respectively), that are quickly configurable using drop and drag processes. In this Fibre Channel model, nodes represent processors, sensors, and displays, and the link is designed to emulate a full speed (1Gbps) Fibre Channel link. The Network Editor allowed rapid construction and test of various possible configurations that might be used in a next generation aircraft architecture.

2. Node Editor

The Node Editor captures the architecture of a network device or system by depicting the flow of data between functional elements, called "modules." Each module can generate, send, and receive packets from other modules to perform its function within the node. Modules typically represent applications, protocol layers, and physical resources, such as buffers, ports, and buses. Modules are assigned process models (developed in the Process Editor) to achieve any required behavior. In this simulation nodes were representative of three types of hardware: Dedicated Mission Processors, Displays and Sensors (which included antennas, radar, Electro-optic and navigation sensors).

3. Process Editor

Each module in a node has an underlying process that defines its behavior. These processes are defined using the Process Editor available in the OPNET Modeler software package. The process is presented to the user as a Finite State Machine (FSM). The FSM is composed of states and pathways. A packet enters the FSM and is acted upon or acts upon the system based on its contents and the state of the system. Tasks can thereby be simple for a low level of abstraction or highly complex making the model a more realistic representation of the actual system.

Inside each of the states is code. This code is C/C++ making modifications easy and understandable to most system developers. Many common tasks, such as finding the value of a field in a packet, setting a value of a packet field etc. are already defined as functions available for use in OPNET. Also available in the Process Editor are Header Blocks, Function Blocks, State Variable Blocks and Temporary Variable Blocks. Header Blocks contain macro definitions, global variable definitions, function prototypes and

transition conditions. Function Blocks contain any user defined functions. These functions are automatically linked to the rest of the code through use of the OPNET compiler. State variables are used to maintain the status of a Node in between process invocations. These variables are created and defined in the State Variable Block. Unlike state variables, temporary variables do not maintain their value in between process invocations. Temporary variables are defined and in some cases initialized in the Temporary Variable Block.

4. Link Editor

The Link Editor allowed the specification of the Fibre Channel physical cabling. This is also known in the Fibre Channel Standard as the Physical Interface or FC-0. By using the Link Editor the Systems Designer is allowed to specify Bandwidth, Bit Error Rate (BER), link cost, propagation delay, packet types supported as well as other attributes of the Link.

5. Packet Editor

The Packet Editor allows the System Designer to "build" packets of the format desired: in this case Fibre Channel Packets. In the Packet Editor one is able to name the sub-parts of a packet, specify the size of that packet (to include variable size data payloads), and specify the type of value accepted by that packet (i.e. integer, float, double, string etc.).

6. PDF Editor

The PDF Editor is a tool that allows the building of user specified Probability Density Functions (PDF). While OPNET Modeler has many predefined PDFs available for use, there are many circumstances where the user would like to use a PDF based either on previous test data or to test a theorem.

B. OPNET MODELER IMPLEMENTATION OF FIBRE CHANNEL

The following section will describe how OPNET Modeler was used to model the behavior of Fibre Channel in an avionics environment. It will describe the method used, the applicability to the Fibre Channel Standard, the data available and the limitations of the model.

The Fibre Channel Model is based largely on the Fibre Channel Standard (FC-PH) [Ref. 12] as interpreted by Robert Kembel in his 1998 book, *Comprehensive Introduction to Fibre Channel.* Due to time constraints, many of the intricacies of the Fibre Channel Standard were not coded. The goal of the model was to have a fairly high level of abstraction, and yet still be able to make meaningful comments on its characteristics such as bandwidth, latency, and upgradeablity. Instances where there are deviations from the standard will be pointed out and their impact assessed. Furthermore the model was designed so that increases in detail and fidelity should be possible by future researchers.

1. The Node

All systems are comprised of nodes. For this model a node will simply be defined as a point from which data originates, is collected, displayed, processed or a combination of all of these elements. Examples of nodes in the system include antennas, mission computers, and displays. Since in this implementation the switch does not originate any of its own traffic or collect/process any other traffic, it will be discussed separately.

In a broad sense any mission avionics system contains only three basic types of nodes: antennas, processors and displays. While there may be more subsystems like navigation, weapons, EW, EO etc, each of the components of these subsystems can be thought of in these terms. It is for this reason that in this model only three types of nodes were created. They are called: FC_ant, FC_disp, and FC_proc. Each node, no matter what type, can represented by four OPNET modules: a packet generator object, a processor, a receiver and a transmitter. Each component of the node will be described in the following sections. The configuration is shown as Figure 7.

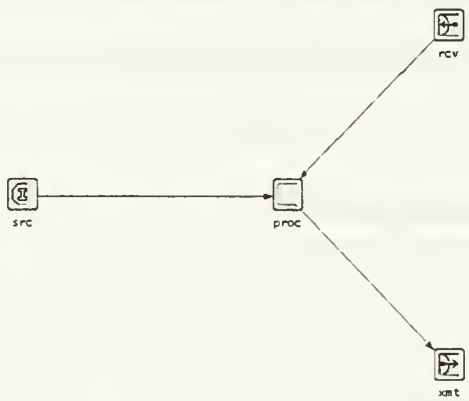

Figure 7. OPNET Implementation of a Fibre Channel Node.

a. Packet Generator Object

All nodes connected to a avionics system will generate message traffic across the Fibre Channel Fabric. Some will produce continuous streams of data (antennas), while others (primarily processors and displays) will absorb this data and generate response and command data. In order to model this behavior The OPNET Modeler Packet Generator Object was used. As described by the OPNET on-line help files, "The packet generator module is a stochastic packet source whose output can be

shaped by user-specified probability distributions. These distributions can control the frequency of packet arrivals and the length of packets." The primary attribute of the Generator that was varied was the interarrival argument. This attribute is in seconds and controls the output bandwidth.

b. The Processor Module

The processor module is the heart of each of the nodes. It contains the FSM (also known as the process model) that defines the behavior of the node. By construction only one FSM was designed. The design was such that based on the state of the node the FSM would take the appropriate path for that state. Since all nodes behave (in a Fibre Channel sense) in approximately the same manner, this design feature has minimal if any impact on simulation fidelity. The FSM and all its states will be examined in detail in Chapter IV.

c. The Transmitter Module

The transmitter is used to forward packets from the node onto the link connecting the node to the switch. There are very few adjustments that the designer has to make when using the OPNET provided transmitter object. The key attributes that have to be changed are the data rate and packets supported. For this model the data rate was set at 1,062,500,000 BPS, which represents full speed Fibre Channel operation. The model was required to support both R_RDY and FC_Header packets. These packets will be described in Section B.4.a and B.4.b.

d. The Receiver Module

The receiver is used to take packets off of the link from the switch and forward them to the node processor. The attributes for the receiver are the same as for the transmitter except that one additional attribute, **ecc threshold**, must be set. **Ecc**

threshold is used to define the highest proportion of bit errors allowed in a packet in order for the packet to be accepted by a receiver and forwarded to an output stream. For this model there was no attempt to model the effects of errors so **ecc threshold** was always set to zero.

2. The Switch

The switch is the device used to connect nodes to each other. This includes connecting switches to other switches. The switch concept is fairly simple. Packets are received, the destination address is noted and the packet is forwarded to the appropriate destination. The switch modeled in this thesis is a 32 port switch. 32 represents the maximum number of links that can be connected to the switch. In a one-switch network 32 nodes could be connected to each other. However when a fabric with multiple switches is created, more than one link can and should be used to connect switches to each other, so less than 32 nodes will be connected to each switch. Much like the node model, the switch is comprised of several basic components, one processor module, 32 transmitter modules, and 32 receiver modules. The switch model is shown in Figure 8.

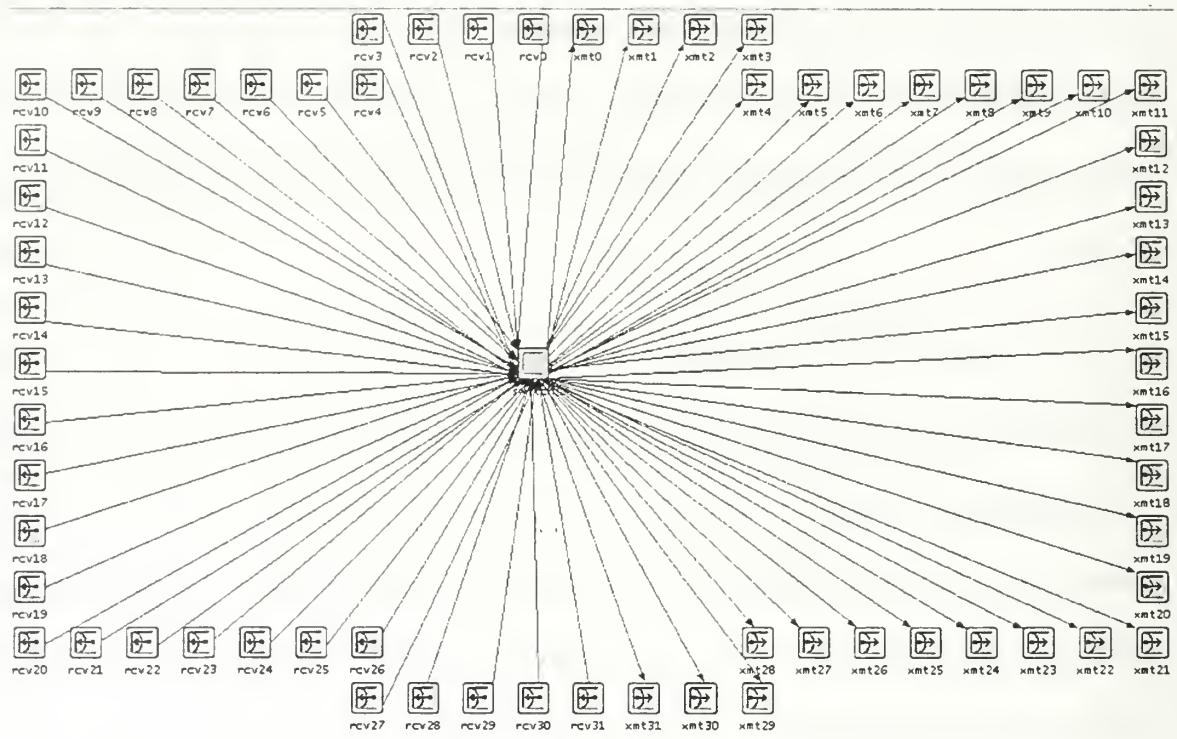

Figure 8. OPNET Model of the Fibre Channel Switch.

a. The Processor Module

Just like the node model, the processor module contains the logic required to receive and forward frames. Contained in the Processor Module is a FSM that defines the behavior of the switch. This FSM and its associated code will be defined in detail in Chapter IV.

b. The Transmitter/Receiver Modules

The 32 transmitter and receiver modules are identical to those used in the node model.

3. The Link Model

All nodes must be connected by links. OPNET has a tool called the Link Editor that allows the designer to build appropriate behaviors into the desired link. For this Fibre Channel model the link was required to have a throughput of 1,062,500,000 BPS and support Fibre Channel frames as well as Fibre Channel primitives.

4. The Packet Model

Each packet sent across the link must be formatted. The Packet Editor was used to make the two types of packets used in this model, Fibre Channel Frames and Fibre Channel Primitives.

a. The Fibre Channel Frame

There are two types of frames in Fibre Channel, Data Frames and Link Control Frames. These are also referred to as FT-1 and FT-0 respectively [Ref. 4]. The format for both FT-1 and FT-0 frames is exactly the same for the first six words (a Fibre Channel word is defined as 32bits before 8b/10b encoding, and 40 bits after 8b/10b encoding). This section is referred to as the Frame Header. The difference between the two frame types comes after the header. Link Control Frames have no data payload whereas the Data Frames can have as much as 528 words of data after the header. Figure 9 shows the OPNET Modeler representation of a Fibre Channel Frame.

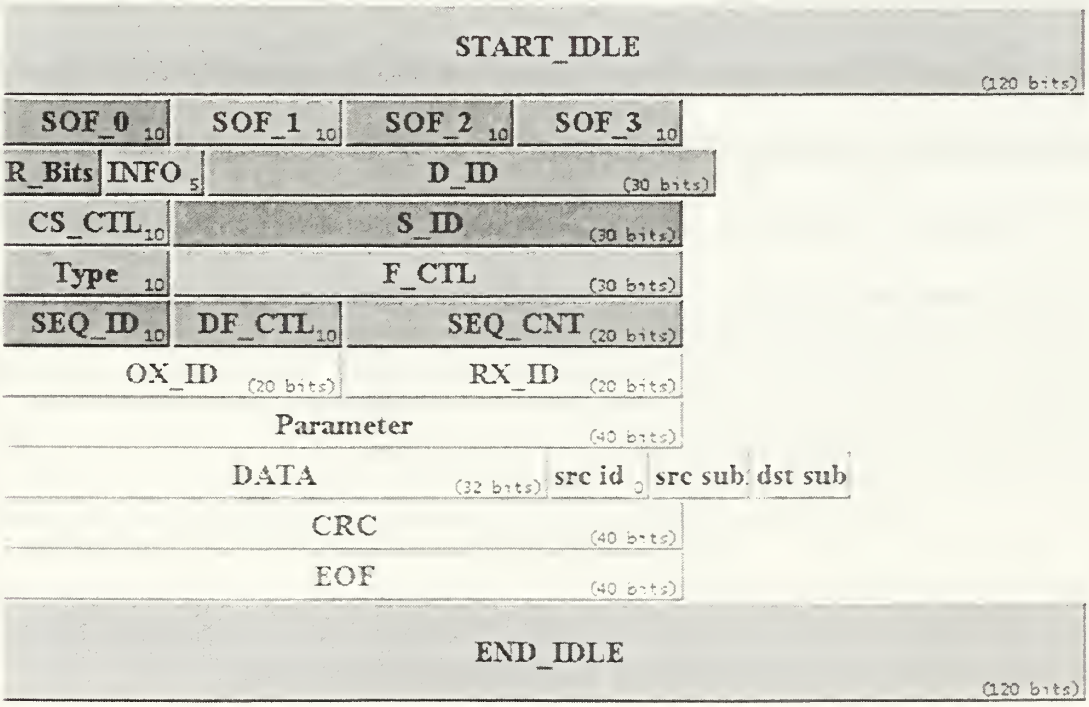

Figure 9. OPNET Representation of a Fibre Channel Frame.

The OPNET packet model is broken up into the following pieces. First, every frame must be preceded and followed by at least three Fibre Channel primitives known as idles. So that bandwidth utilization and latency data would be accurate these 240 bits were added to the front (120 bits) and rear (120 bits) of the frame.

The next piece of the frame is called the Start of Frame or SOF. The SOF identifies the beginning of the frame and contains information as to the Class of the frame that follows, whether to expect one frame or more than one frame as well as other high level information. In this model each frame was considered an independent piece of data and not part of some larger group (One analogy might be that this simulation models people shouting out random words that are not part of a sentence or paragraph.). For this reason the SOF was only used to identify what the class of frame was.

After the SOF the next six words represent the Frame Header. In the model (Figure 9) the header is represented by the packets from **R_Bits** through **Parameter**. The header contains information about the originator of the message, the intended recipient, whether it is a FT-0 or FT-1 as well as many other pieces of data that are required in a full Fibre Channel Protocol. Since many of the features of Fibre Channel were not modeled in this simulation, not all of these fields were used. However, in order to allow future users of this model to increase the fidelity of the model, most of the fields were dutifully represented.

Following the header is the data payload. Though shown in Figure 9 as 32 bits, the data payload is variable from 0 to 528 words. This variability allows for PDF modeling of data payloads based on experimentation/vendor data.

Several additional fields (**src id, src subnet, and dst subnet**) were added to aid in the simulation of some Fibre Channel behaviors. These fields have zero length and therefore do not affect any critical system parameters (specifically, bandwidth utilization and end-to-end delays).

Next in the OPNET Fibre Channel Frame is the Cyclic Redundancy Check (CRC) field. The CRC is used to check the validity of the frame header and data field that were received. Since OPNET has the ability to flag frames with errors and then check for these flags, this field was not explicitly used. It does however affect the size of the frame, and therefore will accurately affect the bandwidth and latency characteristics of the system.

The End of Frame (EOF) delimiter signals to the node that it may end the reception of the frame. The EOF also adds information as to the location of the frame in the larger context of the communication between the nodes. Using the conversation analogy, the EOF delimiter would represent spaces between words, punctuation or the end of a thought. Since this simulation does not encode at higher than the frame level, this capability is not used.

b. The R_RDY Primitive

The second type of packet used in this network is the R_RDY primitive. In Fibre Channel parlance a primitive is a word size (32 bits before 8b/10b encoding, 40 bits after the encoding) packet used to indicate an event at the sending port [Ref. 4]. For this model only one of the primitives is used, the R_RDY. The R_RDY is used for buffer-to-buffer flow control, and indicates that the intended receiver (node or switch) of

a frame has emptied a buffer and is ready to receive another frame. The OPNET representation of the R_RDY is shown in Figure 10.

Figure 10. OPNET Representation of a Fibre Channel Frame.

The **src_nde** field has zero length, and is only used to aid in routing of the R_RDY from its originator to the destination. It has no effect on simulation fidelity.

A future improvement of this model would include the modeling of other primitives and noting their effect on the simulation.

IV. FINITE STATE MACHINE DESCRIPTIONS

A. OVERVIEW

As was discussed earlier, the heart of the OPNET rendition of both the Fibre Channel Node and the Fibre Channel Switch is the Finite State Machine (FSM) contained in the processor module. It contains all the code that defines the behavior of the node/switch. Appendix B contains the code for the OPNET Fibre Channel model, and this chapter will describe each state in detail.

B. THE NODE FINITE STATE MACHINE

The node FSM is shown in Figure 11. The following paragraphs will trace each of the states and describe their function. Wherever deviations from the Fibre Channel Standard are programmed, the rationale behind this decision will be explained as well as what impact this design decision may have on simulation fidelity.

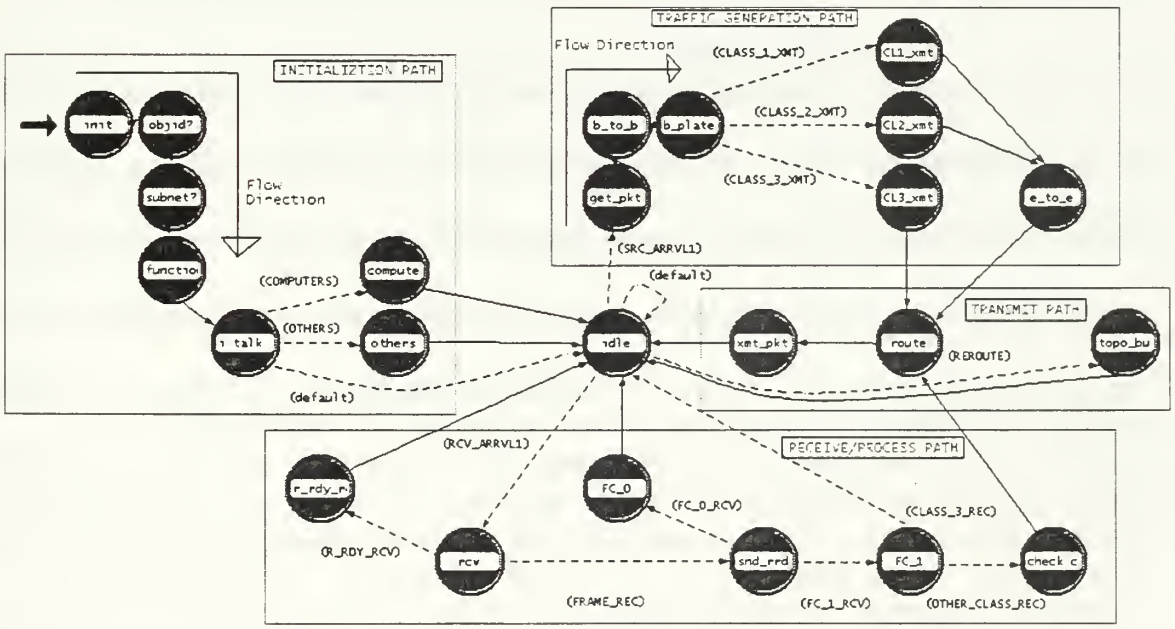

Figure 11. Fibre Channel Node Finite State Machine.

The node FSM is broken up into roughly four functional areas or paths, the initialization path, the packet generation path, the receive/process path, and the transmit path. Some states do no fit neatly into only one path. These states were put into the functional area that seemed to be the best fit.

1. Initialization Path

During initialization the node finds out who it is and what other nodes are in the network. Initialization takes place prior to any communications in the model. Several OPNET "housekeeping" chores are taken care of. These include the declaring of global variables, scheduling of interrupts, declaration and initialization of statistics, and initialization of the state of the node. The initialization path is composed of the following seven states: **init**, **objid?**, **subnet?**, **function?**, **i talk to**, **computers**, and **others**.

a. Init State

The **init** state is primarily used to initialize variables, register statistics set up the administration of the process. It does not affect the fidelity of the model.

b. Objid? State

In Fibre Channel there are levels of login: Fabric Login (FLOGI), Port Login (PLOGI) and Process Login (PRLI). In this model there are no upper level processes (FC-4). All communication occurs at the FC-2 level and below. For this reason PRLI is not addressed in the model. FLOGI is used by one node to establish communications with the fabric. PLOGI is used for one node to establish communications with another, specific node. Fibre Channel allows FLOGI and PLOGI to be done via explicit or implicit means. [Ref. 4] In this model each node learns the state of the fabric and the other nodes in the fabric via implicit means. That is to say, there are

no explicit Fibre Channel communications between nodes prior to passing of meaningful traffic. In order to facilitate this, each node must learn its object id or **objid**.

The **objid** is a unique integer value given to all objects in the model at run time. It is a software attribute applicable to any OPNET Modeler model and has no parallel in Fibre Channel. Since each node address in the fabric must be unique, the **objid** was used to satisfy this requirement. The **objid?** state was used by the node to identify its own **objid** as well as to identify some of the attributes of the node. These attributes are Subsystem, Process Delay, and R_RDY Delay. The Subsystem attribute identifies of what subsystem the node is a part. Subsystems include Radar, EW, EO, NAV and Navigation. Other subsystems could easily be added to more closely model existing or planned avionics architectures. Process Delay is used to model the time it takes for a processor to read, analyze and reply to a frame. R_RDY Delay is used to model the time required for a node to return an R_RDY.

c. Subnet? State

In order to route in OPNET both an objid and subnet are required. The **subnet?** state is used to determine the subnet of the node. This is an administrative state only and is only required so that the OPNET Modeler routing functions may be used. It neither adds nor subtracts from the fidelity of the simulation.

d. Function? State

There are several ways one could choose to design a distributed architecture. One way would be to have all processors available to run any task. Sensors would send their data for processing based solely on processor load. This has several drawbacks. First, processors would continuously have to load and unload the required application software in order to process the data required, thereby slowing down

processing time. Second, some account must be made for the cost of transporting the data through the fabric. It could take much longer to send data to a light loaded processor 4 or 5 hops away than to send it to a processor located on the same switch.

Another way to design the architecture would be to have the processors and sensors grouped according to function. This would allow the software to remain "close by" in high speed RAM. The advantage of total distribution would still be there since any node in a fabric could talk to any other node provided that a redundant pathway exists. This would allow, for example, a primarily NAV based computer to process radar data in the event of battle damage. The fabric would simply have to route the required radar software and radar data to a NAV processor.

This simulation was designed using the second philosophy. The **function?** state discovers the function and associated subsystem of the node. Since all the nodes have a standard "type name" (i.e. FC_ant, FC_proc, FC_disp), a string compare command was used to compare the "type" attribute with the standard names. A similar process was used to discover the node's subsystem.

e. I Talk To State

The **i talk to** state is used to help the node discover what other nodes it will have to communicate with. A description of the logic follows. If the node was of type "FC_ant" the node is some kind of sensor and it could only be expected to talk to a processor. The assumption was make that sensors did not talk to other sensors except through an intermediate processor. Likewise, if the node was of type "FC_disp" it could only be expected to communicate with display processors. Lastly, if the node was of type "FC_proc" it could reasonably expect to talk to other processors, displays and sensors.

f. Computers State

The **computers** state is one of two states (**others** is the other one) which takes all the information gathered so far and builds a look up table of all the other nodes with which this node will communicate. During this state the node also gets other essential information about these nodes. Specifically, the amount of End-to-End credit available for use. These values are stored in state variable arrays. These variables are then available for use and modification during each process invocation. This state essentially fills the need for an implicit login.

g. Others State

The **others** state is nearly identical to the **computers state**, but is used when the node of interest is a processor. The reason for this is that an EW computer could very well have reason to communicate with the display group of processors. This state allows for processors from one subsystem to communicate with processors from other subsystems. It does not however allow a processor from one subsystem to talk to a sensor from another subsystem.

The design does have some impact on the fidelity of the simulation. Currently there is no way to update the communication list in the event of battle damage. The consequence of this is that if, for example all of the radar processors were destroyed there would be no way (in the simulation) to divert the data to another processor.

2. Traffic Generation Path

The node uses the traffic generation path to generate its own traffic. This path is invoked when an interrupt from the ideal packet generator is received. Eight states are grouped into the traffic generation path. They are **idle**, **get_pkt**, **b_to_b**, **b_plate**, **CL1_xmt**, **CL2_xmt**, **CL3_xmt**, and **e_to_e**.

a. Idle State

The **idle** state could easily be thought of in either the transmit or receive path. It is simply a state from which the process waits to be invoked. The idle state represents a node waiting to receive or send traffic.

b. Get_pkt State

In the node model packets come from one of two places, either the packet generator or the receiver. When the packet comes from the packet generator it must be assembled, a destination assigned, routed and sent. In this model the first step is the assigning of a destination. to accomplish this, the **get_pkt** state picks a uniformly distributed integer, which is then used as an index in the array of suitable destinations that was built in the **computers** or **others** state. A uniform PDF was chosen because the assumption was made for this model that communication with other nodes in subsystem group was equally likely. This distribution could be changed to fit any vendor/real-system data.

c. B_to_b State

In Fibre Channel there are two types of flow control, Buffer-to-Buffer and End-to-End. Both types are based on an incrementing/decrementing credit system. Each node has credit with other nodes in the system. These values were obtained during the implicit login that occurred in the computers or **others** state. Buffer-to-Buffer credit is designed to ensure that the next node in the routing through the fabric (in a fabric this will always be a switch) has enough space in memory/buffers to accept the frame.

The **b_to_b** state is designed to check if there is buffer-to-buffer credit available. In reality if there is no credit available the frame must not be sent. It must somehow either be stored until credit is available (credit replenishment will be discussed

later) or destroyed. Currently the login Buffer-to-Buffer credit value is known implicitly by the node. In future designs it may be useful to discover this value based on the switch that is attached to the node of interest. This would allow the system designer/evaluator to place switches with known memory configurations in the system and evaluate the impact of delays and throughput.

Since the purpose of this model is to evaluate designs, all packets are sent regardless of credit available, but the **b_to_b** state logs each time a packet is sent without credit and the credit is then replenished to its login value. This node statistic, known as "no_bb_count", is then available to the system designer/evaluator. This deviation from the standard was not considered to have a significant impact on simulation fidelity.

d. B_plate State

There is a certain amount of similarity between frames in Fibre Channel. Much of the header field is common between frames originated in the same fabric. In the Navy things that are common and can be done in advance of any real work are called "boilerplate", hence the name of the state.

The state **b_plate** simply sets some of the standard fields in the header of any frame that is sent. The values are taken from the standard as described on pages 179-206 of Robert Kembel's book [Ref. 4]. Also, in this state the destination and source addresses are inserted in the frame header. These values were obtained in the **objid?** state.

As a side note, there was a provision made to change the size of the data payload associated with each data frame. This function was rarely used during the test and analysis of various systems. The reasons are as follows. First, no effort was made to

model the Sequences and Exchanges associated with Fibre Channel. By using this approach, each node could be viewed as a frame-generating machine. Since the data field in Fibre Channel is relatively small (2.0625 Kbytes), it was assumed that most data being passed in any exchange would be greater than this (i.e. would require at least one, but probably more full-length frames). The result of this would be that the average data payload would approach the limit of 2.0625 Kbytes. After an analysis of a system it would be simple to vary the size of the data payload based on any PDF. This model and all data associated with it represent a worst case scenario.

e. CL1_xmt State

The **CL1_xmt** state is used to fill in fields that uniquely identify the frame as a Class 1 message.

f. CL2_xmt State

The **CL2_xmt** state is used to fill in fields that uniquely identify the frame as a Class 1 message.

g. CL3_xmt State

The **CL3_xmt** state is used to fill in fields that uniquely identify the frame as a Class 1 message.

h. E_to_e State

End-to-End credit is only applicable to Class 1, 2 and 6 frames. Since there is no delivery assurance in Class 3, only the Buffer-to-Buffer flow control criteria is required to be met prior to data transmission. The **e_to_e** state was designed to check for the availability of End-to-End credit prior to data transmission.

The **e_to_e** state operates in a manner very similar to the **b_to_b** state in that all frames are sent regardless of the availability of credit. Once again, when the credit allocated at login is depleted a node statistic called "no_ee_count" is incremented

and the End-to-End credit is replenished. As with the **b_to_b** state, the value added by this statistic was deemed to more than offset the loss in simulation fidelity.

3. Receive and Process Path

The receive and process path is invoked upon receipt of a packet from the node's receiver. This packet could be an R_RDY primitive, an FT-0 or FT-1 frame. The receive and process path is comprised of six states. They are the **rcv**, **r_rdy_rcv**, **snd_rrdy**, **FC_0**, **FC_1** and **check credit** states.

a. Rcv State

The **rcv** state is responsible for determining whether the incoming packet is a Fibre Channel frame or a primitive. Since the only primitive defined for this model is the R_RDY, this state determines if the received packet is a frame or an R_RDY. This task is accomplished by checking the number of fields in the incoming packet. Primitives only have one 40-bit field whereas frames have many more.

b. R_rdy_rcv State

The FSM associated with the node processor has to deal with two types of frames: frames that are produced by the node and frames that are received and processed by the node. In order to replenish the supply of Buffer-to-Buffer credit the system relies on receipt of a Fibre Channel primitive known as an R_RDY. Each time a switch receives any valid frame it returns an R_RDY to the node that forwarded the frame to it. Then, when the node receives the R_RDY, the Buffer-to-Buffer credit is incremented.

The **r_rdy_rec** state is responsible for incrementing the buffer if it is less than full. If the buffer is already at its login value the buffer is left at the login value. The standard does not allow Buffer-to-Buffer credit to increase beyond its login value.

c. Snd_rrdy State

If the node receives a frame from the switch it replies with an R_RDY primitive. The **snd_rrdy** state is responsible for constructing, routing and calculating the delay associated with sending the R_RDY. R_RDYs are sent immediately upon receipt of any frame, valid or not. The receipt of an R_RDY only indicates to the receiver that the resource or buffer is once again available to receive another frame. For this reason the R_RDY delay would be considerably faster than processing delays. Once again, an accurate simulation of a system would require the PDF of this delay to be defined. Currently an either an exponential PDF is used with the node attribute "R_RDY Delay" as the mean of the PDF, or the code is slightly modified to allow constant delays to be modeled.

d. FC_0 State

The **FC_0** state is entered if the received node is a LCF (FT-0) frame. Several tasks are accomplished. First the End-to-End credit associated with the sending node is incremented. Next, the time required to send the frame and receive the ACK is calculated and logged. This is a round trip time. Round trip time is used instead of one way in Class 1, 2 and 6 communication because that is the time required to ensure that the data reached its intended recipient. If only one time was used as in the case of Class 3, it would only represent the time required to send and receive a frame of unknown validity. Using this method ensures that if an ACK is received to a frame, the intended recipient did not only receive it, but also that it was valid and that task is complete.

Another use of the **FC_0** state is to determine if the LCF was a n F_BSY instead of the expected ACK. When that occurs at least one of the switches on the path to the destination node or the destination node itself was either busy with Class 1 or 6

communications or did not have adequate resources to accept the frame. Each time an F_BSY is received a statistic "F_BSY_DAT" or "F_BSY_LCF" is recorded. This statistic helps analyze if adequate memory resources are available along various pathways.

There are two types of F_BSY LCFs. The first type is in response to a FT-1 the other is in response to an FT-0. The purpose for differentiating between the two is to ensure that if a data frame was sent that the transmitting node is made aware that the frame did not reach its intended recipient. In order to allow flexibility to the system designer, Fibre Channel does not require a retransmit of the data. Secondly, so that endless "I'm busy to your busy of my busy etc. etc. etc" do no occur, F_BSYs are not returned when the frame that could not be sent was an F_BSY.

The last function of the **FC_0** state is to destroy the packet. This destruction is a key step in OPNET Modeler because it frees all the memory resources associated with that frame. Failure to destroy the packets slows down simulation and can cause the system to crash because of memory leaks.

e. FC_1 State

In Fibre Channel there are two types of frames defined. The first is Frame Type 0 or FT-0 which is a Link Control Frame (LCF). LCFs have a data length of zero and are used for administrative tasks. Frame Type 1 or FT-1 frames are data frames. During construction of the model a small typographical error was made and FC-1 was used in place of FT-1. This error is prevalent throughout the model.

Upon receipt of a data frame the node uses the **FC_1** state. If the incoming data frame is a Class 3 message, there is no need to assure the sender that the

frame was received, and the only tasks accomplished here are of a statistical nature. The time it took for the frame to transmit the fabric is recorded, and the frame is destroyed.

If the incoming data frame is a Class 1, 2 or 6 frame a response from the node to the sender is required to assure the sender that the frame was received in tact. This is done through use of one of the FT-0 frame types called an ACK. In the **FC_1** state the ACK is constructed, and prepared for routing.

f. Check Credit State

When it is required to return an ACK, the state **check credit** simply checks the value of the node's Buffer-to-Buffer credit prior to sending the ACK back. If there is Buffer-to-Buffer credit available, it is decremented. If, however, it is zero it is treated as in the **b_to_b** state. In its value is replenished to the login value and the "no_bb_count" statistic is incremented.

4. Transmit Path

The transmit path is designed to calculate routes, update the network topology and send the packets out the receiver. Three states are included in this path. They are topo_build, route and xmt_pkt.

a. Topo_build State

In order to route frames through the fabric, OPNET Modeler must have a definition of the associated fabric or topology. The **topo_build** state is used to build and rebuild the topology of the fabric. This topology is stored as a global pointer available to all nodes and switches in the fabric. This pointer contains information such as the cost of transiting each link, all object ids and subnets in the fabric.

The function required to "build" the topology is very expensive computationally. In order to minimize simulation time and reduce memory usage. A

decision was made to rebuild the topology on a scheduled basis. This schedule is based solely on simulation time and not linked to any packet arrival at the processor. In this respect it should be considered an asynchronous event. This has some fidelity ramifications. First, the optimum path between two nodes is based on this topology pointer and the information associated with it. This fact means that between rebuilds of the topology the frame will take a less than optimum path. This is not unlike many systems in use today. Second, if nodes are disabled between topology rebuilds, frames may start out for a destination that is working when the frame leaves its source, but disabled by the time it arrives. The solution to increased levels of fidelity is to rebuild the topology more often with the limit being rebuilding the topology before each frame is transmitted. The tradeoff is increased memory usage and decreased simulation speed.

b. Route State

Any fabric has to have a algorithm to route frames from the source node to the destination node. Since the speed and efficiency of a system are normally of paramount concern to the user, and one measure of what makes one system "better" than another, most routing algorithms are proprietary in nature. This being the case, the routing algorithm used in this simulation was the default algorithm based in OPNET Modeler. The **route** state was designed to add a route pointer to the frame of interest. The routing algorithm was based solely on the cost of getting from source to destination. Cost was defined as a function of bandwidth usage. The higher the usage the higher the cost to transverse that link. By using this cost definition, bandwidth could be shared to the maximum extent possible.

Like topology building, route building is computationally expensive, and should be minimized. To that end, **route** was designed to first "check and see" if a route to the destination had already been created. If it had been created that route pointer was used. If the route did not exist it was created and added to the state variable array that contained route pointers for that node. This way the route pointer would be available for later use and no unnecessary routes would have to be built.

c. Xmt_pkt State

The processing of commands in OPNET Modeler adds no time to the simulation time. Time is only added by specific calls of the OPNET function "op_pk_send_delayed." In reality for any node to process incoming frames takes finite time. This time or processing delay is simulated through use of the "op_pk_send_delayed" function. Current implementation uses an exponential PDF as to model these delays. However, with vendor data better models of this delay could be implemented, further enhancing the fidelity of the model.

C. THE SWITCH FINITE STATE MACHINE

The switch in Fibre Channel is the heart of what has been referred to as the fabric configuration in Fibre Channel. In this model the switch was viewed in fairly simplistic terms. A switch's job is first to receive frames from one node or switch. Next it must accomplish any administrative overhead associated with the protocol. For this Fibre Channel model these tasks are mostly limited to sending R_RDYs back to the sender in order to ensure flow control. Lastly, it must transmit the frame to the next switch on the route or to the final destination. For these reasons it was modeled as 32 pairs of transmitters and receivers configured around a processor. It is described as a 32-port switch. Like the node, the processor is modeled by a FSM. Like the node FSM the

switch FSM is broken up in to several areas that correspond roughly to their functionality. These are the Initialization Path, the Process Path, and the Transmit Path. The switch FSM is shown in Figure 12. Appendix C contains the OPNET code for the switch FSM.

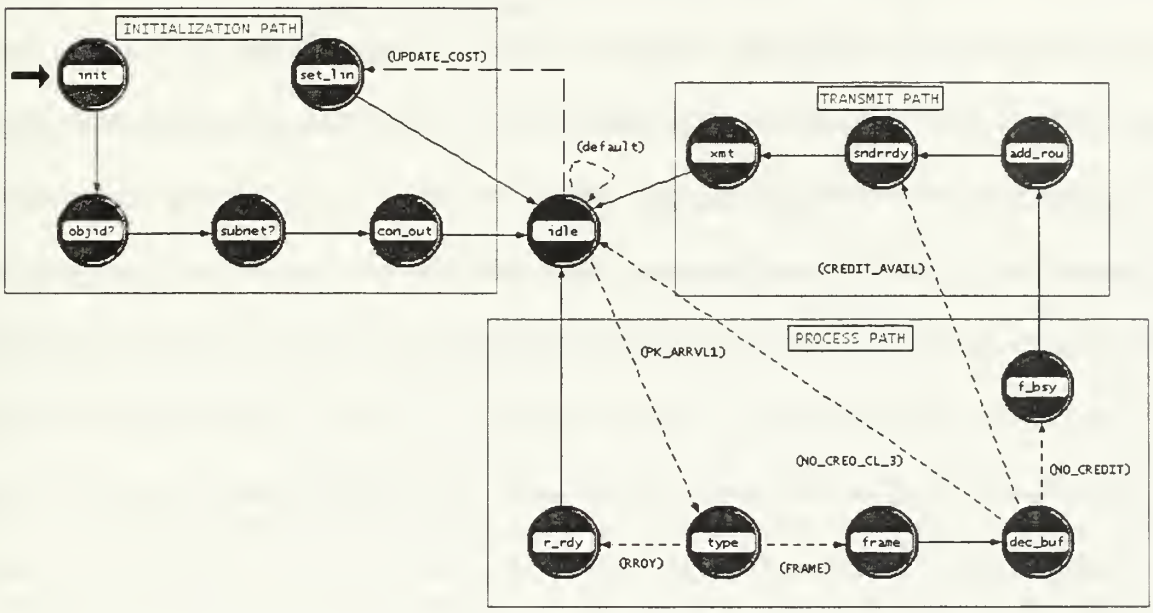

Figure 12. Fibre Channel Switch Finite State Machine.

1. Initialization Path

Just as with the node several administrative tasks have to accomplished in the switch before it can process any network traffic. This accounts for procedures that would take place during an explicit or implicit login and contains five states, **init**, **objid?**, **subnet?**, **con_out**, and **set_link_cost**.

a. Init State

During the **init** state three functions are accomplished. First the Buffer-to-Buffer credits are set to the login value. Second, utilization for the links is set to zero. Utilization is used in the routing of frames. In order to ensure that the bandwidth across

all redundant links is balanced, frames are sent on the route with the lowest utilization. Lastly, global statistics are initialized and the first interrupt used to set link cost is scheduled.

b. Objid? State

The **objid?** state in the switch model functions almost exactly as the **objid?** state in the node model. The switch is given a unique ID and the values for mean processing delay and mean R_RDY delay are set. In an abstract sense a switch simply routes frames around the fabric. This routing takes finite time. The two delays can be thought of in the following manner. The mean time it takes to recognize that the incoming signal is a frame plus the time required to construct and send the R_RDY primitive back to the previous node/switch equals the mean R_RDY delay. The mean time it takes to analyze that frame and determine how big it is where it is going, find out if that resource is available (through flow control), how to get it there, and finally to send it is what is defined in this model as the Processing Delay.

c. Subnet? State

In order to route in OPNET both an objid and subnet are required. The **subnet?** state is used to determine the subnet of the node. This is an administrative state only and is only required so that the OPNET Modeler routing functions may be used. It neither adds nor subtracts from the fidelity of the simulation.

d. Con_out State

The state **con_out** is designed to find all links attached to the switch and build a look-up table, so that when a packet is routed to a node/switch the transmitter attached to that node/switch can be found quickly. This state has no impact on the fidelity of the simulation. It does, however, affect the speed and resources required in

running the simulation. The OPNET functions required to build the arrays of attached Fibre Channel link models, nodes and switches are very expensive computationally. If these functions had to be called each time a frame was forwarded, simulation run time would become inordinately long, rendering the simulation all but useless.

e. Set_link_cost State

In order to ensure balanced bandwidth utilization across redundant links, cost of traversing a link had to be a function of bandwidth utilization. The **set_link_cost** state is an asynchronous state (much like the **topo_build** state in the node FSM) that computes the utilization of each link and applies it on a scheduled basis. The more often this state is called the more accurate the simulation. The price paid is in simulation speed and memory resources.

2. Process Path

The Process Path is that portion of the switch FSM that looks at the received packet (primitive or frame), analyzes it, and decides what actions must be taken in response to that packet. This path contains the following states, **idle**, **type**, **r_rdy**, **frame**, and **dec_buf**.

a. Idle State

The **idle** state could easily be thought of in either the process or transmit path. It is simply a state from which the process waits to be invoked. The idle state represents a switch waiting to receive or forward traffic.

b. Type State

This state assesses the type of packet received at the switch receiver. To do this task the state counts the number of fields in the packet received. For any primitive there are less than three fields, whereas for a Fibre Channel frame there are many more.

c. R_rdy State

In this model, if a packet received has less than three fields it has to be a Fibre Channel primitive. Since only R_RDY primitives are supported in this model it has to be an R_RDY. Upon receipt of an R_RDY this state determines which node sent the R_RDY and increments the buffer-to-buffer credit for that node. Since the credit amount can never be greater than the initialized credit value, a mechanism was put in place to prevent credit from exceeding that value.

d. Frame State

Each packet transmitted in the OPNET simulation has an associated route pointer. This pointer references the route that the packet will take from its source to its final destination. If the switch receives a frame the **frame** state advances the route pointer so that the frame is directed to the next node in the route.

e. Dec_buf State

When a frame reaches the **dec_buf** state some determination must be made as to the status of the switch's buffer-to-buffer credit with the next node in the route. The **dec_buf** state checks to see if buffer-to-buffer credit is available, and then takes actions based on that information.

If the frame is Class 3 and no credit is available with the next node the frame is destroyed and the FSM transitions back to the **idle** state and awaits the arrival of the next packet. The additional step of incrementing the "Dead Class 3" statistic is completed here to aid the system designer in system optimization. If the frame is Class 2 and no credit is available with the next node the FSM transitions to the **f_bsy** state.

f. F_bsy State

If a Class 2 frame (either FT-1 Data frames or FT-0 Link Control Frames (LCF)) cannot be forwarded to the next node in the route, some notification must be

made to the originator of the frame. The **f_bsy** state accomplishes this goal by turning the frame into an F_BSY LCF and returning it to the originator of the frame.

3. Transmit Path

The Transmit Path in the switch consists of three states, **add_route**, **sndrrdy**, and **xmt**.

a. Add_route State

When there is not enough buffer-to-buffer credit available to forward a packet and an F_BSY must be returned to the originator of the message, the switch is forced to add a route to the F_BSY from the switch to the originator. The **add_route** state computes a minimum cost route from the switch to the originator of the frame and adds that route pointer to the packet.

b. Sndrrdy State

The **sndrrdy** state constructs, routes, and sends with some user defined delay the R_RDY in response to the frame received. Current implementation allows for constant delays in the switch, however PDF based delays could easily be added to increase simulation fidelity.

c. Xmt State

The **xmt** state serves two functions. First it adjusts the utilization of the link which the packet is being sent out on. Since the routing in this switch uses a minimum cost algorithm to route packets, and cost is a direct function of utilization, this state updates the utilization for each packet sent out of the switch. Second, this state adds any delay associated with the processing of the frame. In the current implementation this is a constant, but for better fidelity it could be modeled by a PDF using vendor supplied data or test output.

THIS PAGE INTENTIONALLY LEFT BLANK

V. MODEL VALIDATION

A. OVERVIEW

In order to ensure that models built with the simulation tool give accurate results some validation process must take place that ensures that the model represents what happens in the physical article. Several tests were conducted using the Fibre Channel model that was designed in OPNET Modeler in order to ensure that any conclusions drawn from the model could be considered accurate.

B. TESTS

1. Throughput

The first test involved finding the functional relationship between the interarrival argument of the Packet Generator Object and the actual throughput of the link. By knowing this relationship actual objects can be modeled with various throughputs simply by changing the value of the Packet Generator Object's interarrival argument.

a. Theory

Throughput will be defined in bits/sec. Interarrival argument defines how often a new packet (or frame in the case of Fibre Channel) is sent from the Packet Generator Object. Interarrival argument is represented in frames/sec. Using dimensional analysis it can be shown that by multiplying $\frac{bits}{frame} \times \frac{frames}{\sec} = \frac{bits}{\sec}$. Since in all likelihood the system designer will have a target throughput of a node and will want to solve for the interarrival argument. In this case the format would be $\frac{\sec}{bit} \times \frac{bits}{frame} = \frac{\sec}{frame}$. As an example if a frame is composed of 20,000 bits and the desired throughput is 5,000,000 bits/sec the interarrival argument would be 0.004.

b. Model

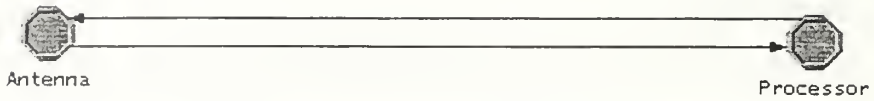

Figure 13. Model Used to Verify Throughput and Overhead.

The model to test the throughput was a simple connection of a processor node connected to an antenna node. This configuration is shown in Figure 13 above.

c. Test and Test Conditions

Since there were only two nodes in the system, all traffic was forced to the other node. This would ensure that all bandwidth utilized was a function of the node of interest. In Fibre Channel the longest frame possible is 21720 bits. The breakdown of one data frame of maximum length is broken down in Table 4.

Table 4. Frame Size Breakdown

Field	**Number of Bits Required for Transmission**
Start of Frame (SOF)	40 Bits
Frame Header	240 Bits
Cyclic Redundancy Check (CRC)	40 Bits
End of Frame (EOF)	40 Bits
Required Inter-Frame Bits	240 Bits
Maximum Size Data Payload	21120 Bits
Total Size	21720 Bits

4.0msec was chosen as the interarrival argument. This equates to a throughput of approximately 5Mbytes/sec (5,430,000 Bits Per Second (BPS) to be exact). Since each frame sent will require the receiver to send an R_RDY in reply to all frames and in the case of Class 2 frames an ACK frame. This would corrupt the data, so to ensure that the data recorded was only due to one of the nodes only the antenna would transmit at the desired data rate. The processor would wait to transmit its first frame until after the time of interest. Also, only Class 3 frames were sent so that no ACK frames were created.

d. Results

The simulation was run for 10 seconds and the results are shown in Figure14 below.

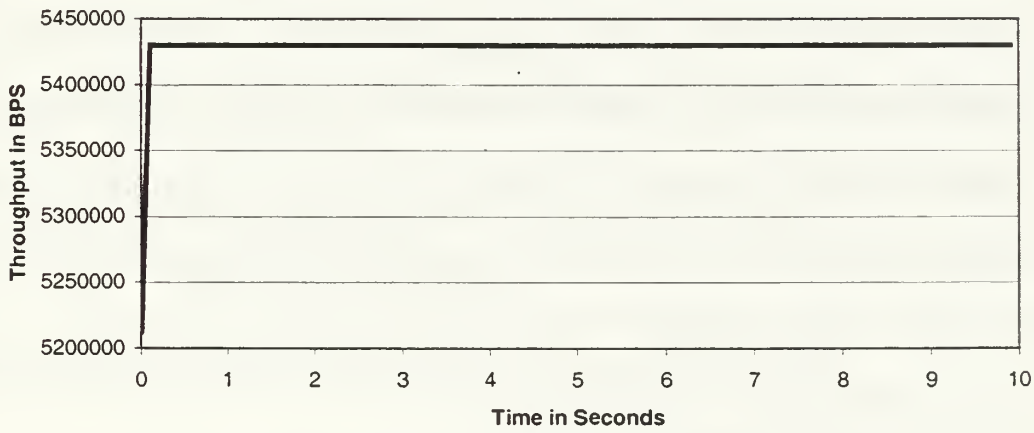

Figure 14. Results of the Simple Throughput Validation Test.

The chart clearly shows a throughput of 5,430,000BPS, which is exactly what was predicted.

2. Overhead

Some method was needed to verify that for every frame sent across the network the proper amount of return overhead was returned. That is to say that for any Class

frame a R_RDY was returned, and for Class 2 service not only was the R_RDY sent, but also an ACK was sent.

a. Theory

An R_RDY represents 40 bits. If only Class 3 messages are sent, and the receiving node does not originate any data traffic, then for each data frame received 40 bits are returned. So, for Class 3 the formula used to verify overhead is exactly the same as the formula used to verify throughput, and that is $\frac{frames}{\sec} \times \frac{bits}{frame} = \frac{bits}{\sec ond}$. The only difference is that in the Class 3 example the return "Frame" is only 40 bits long.

For Class 2 messages it was assumed that for every frame sent an ACK was returned as well as the R_RDY. This adds overhead on the return path. The basic formula remains the same, but to get the total BPS of overhead one must add the overhead due to the R_RDY and the overhead due to the ACK. When using the ACK_N method (described in chapter 14 of Ref. 4), which only returns one ACK for N data frames, the overhead will be reduced. This model does not support the ACK_N format.

b. Model

The same processor and antenna model used in the previous test (Figure 13) was used in the Class 2 overhead test.

c. Test and Test Conditions

The simple system shown in Figure 13 was configured so that only the antenna would originate data. This would ensure that all bits traveling on the return path from the processor would represent overhead traffic in response to the frame sent by the antenna. An interarrival argument of 0.004 seconds was again used giving a throughput from the Antenna to the Processor of 5,430,000 BPS. The test was conducted first using only Class 3 traffic and then using only Class 2 traffic.

d. Class 3 Traffic Results

For Class 3 traffic the expected overhead is due only to the 40 bit R_RDY primitives being returned to the antenna for buffer-to-buffer flow control. The theoretical value would be 40/0.004 or 10,000 BPS.

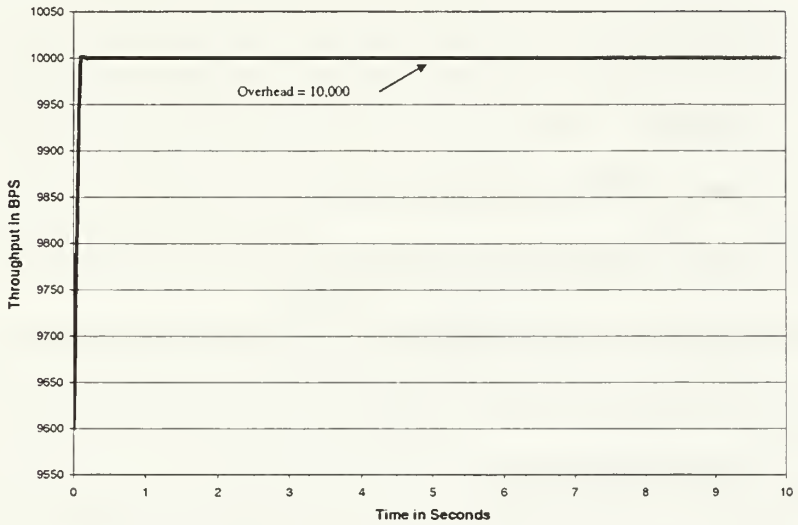

Figure 15. Results of the Class 3 Overhead Verification Test.

Figure 15 above shows that indeed the model accurately represents this.

d. Class 2 Traffic Results

For Class 2 traffic the only additional overhead is due to the ACK response to the message. Since the ACK is a 600-bit frame, the throughput would be equal to 600/0.004 or 150,000 BPS. This added to the 10,000 BPS caused by the R_RDY primitive results in a predicted throughput of 160,000 BPS. Figure 16 shows that the throughput on the return path from the processor to the antenna was exactly 160,000 BPS thus verifying the model's behavior.

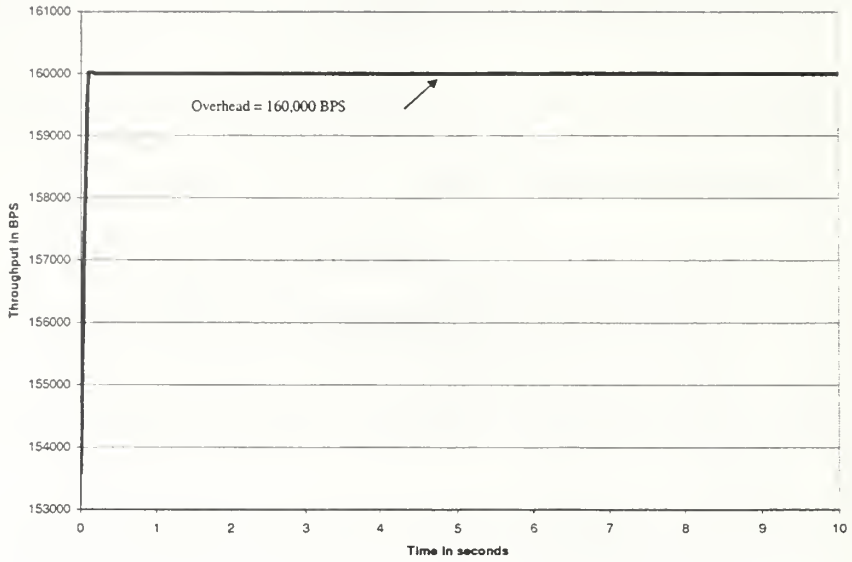

Figure 16. Results of the Class 2 Overhead Verification Test.

3. End-to-End Delay

The next test was conducted to verify that end-to-end delays inherent in Fibre Channel were modeled correctly. The tests involved two models. One was a long distance two-node network similar to the one used in the Throughput and Overhead tests. This model was used to verify that the propagation speed through the fiber was accurately modeled. The second was a was a short distance, two-node, two-switch network with aircraft like representative distances involved.

a. Theory

It takes time for bits to travel across a medium. Even in a vacuum nothing travels faster than 3.0E+08 m/sec. The time that elapses from the sending of the first bit to the reception of the last bit is the delay. Having predictable delays is the key to having a deterministic system. One analogy involves the train schedule. If the train arrives at the station at the same time every day it is easy to know when to be at the train station to catch the train. If, however, the train never leaves or arrives at the same time, it is very

difficult to make any decisions based on what amounts to a meaningless schedule. Knowing the time frame when frames will arrive after being sent is key to the system designer.

The network modeled is a full speed Fibre Channel network using optical fiber as the medium. The full speed network operates at 1,062,500,000 BPS. At this rate it takes 0.94117647nsec to transmit one bit. At this rate a full data frame of 21,720 bits takes 20.44235usec to transmit. This delay represents the largest uncontrollable delay in the system. The only other uncontrollable factor is the geometry of the aircraft, which in turn drives the length of the interconnecting optical fiber. In a fighter/military aircraft wire runs of greater than 35 meters are unlikely. Given that, and assuming a propagation speed of 200,000,000 m/sec (based on an index of refraction of 1.5, and the speed of light as 300,000,000 m/sec) the maximum delay caused by the medium would be 0.175usec, or two orders of magnitude smaller than the transmission delay. All other delays are in some manner controllable by the system designer. It was this factor that drove the requirement to verify that end-to-end delay was a function of transmission delay plus the delay caused by the medium.

For the purposes of this simulation two types of end-to-end delay were considered. First, when a Class 3 message is sent there is no acknowledgement so the end-to-end delay is clearly the difference between the time when the first bit is transmitted and the time when the last bit is received. For Class 2 a distinction was made. Since all in this model all Class 2 frames require an ACK (ACK_N is not supported), the end-to-end delay was calculated to be the difference between the time

when the first bit is transmitted and the time when the last bit of the returning ACK is received.

b. Long Distance Model

The first model designed was used to ensure that the propagation speed of the medium was properly measured. In order to do this a network was set up that was a known distance, in this case 10,000 meters. Only two nodes were involved. The model is shown in Figure 17.

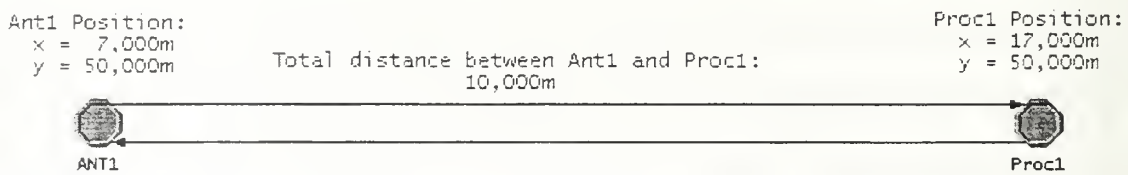

Figure 17. Long Distance Model.

c. Test and Test Conditions

This configuration was tested in two modes the first was using only Class 3 messages the second using only Class 2 messages. In order to keep the data uncorrupted the antenna was configured to transmit 100 full length (21,720 bits) data frames/second while the processor was confined to only responding to the antennas traffic and generating none of its own traffic. Neither the processor nor the antenna added any delay to the system. While this is physically impossible, this constraint was added so that the end-to-end delay would only be a function of transmission delays and delays due to the propagation through the media. For the first test one packet would take 20.4423usec to transmit and 50usec to propagate through the fiber. Therefore the expected end-to-end delay would be 70.4423usec. Figure 18 shows the results of the test.

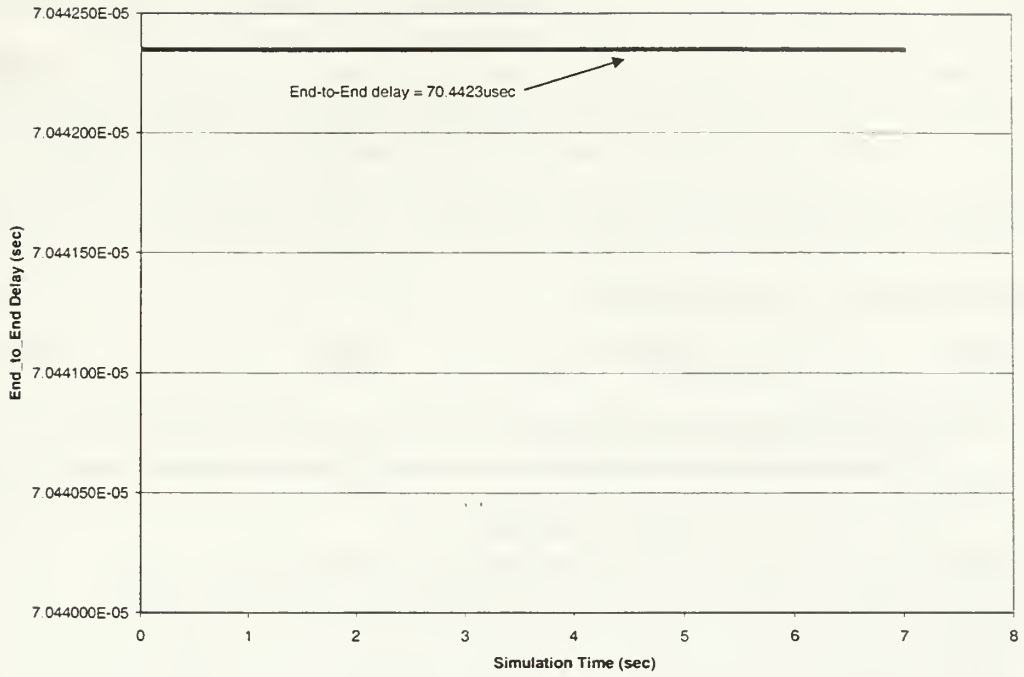

Figure 18. Results of the Class 3 End-to-End Verification Test.

For the second test all the frames that were sent were Class 2. For this test the end-to-end delay would be expected to be greater than for the Class 3 case. Formally the delay should be equal to two times the propagation delay (100.0usec) plus the transmission delay for one full length data frame (20.4423usec) plus the transmission delay for one ACK frame (0.56471usec). The sum of these would be 121.007059usec. During the test the actual value was found to be 121.0447usec, or a difference of 37.6412nsec. It was determined during the model validation that this delay was equal to the time to transmit 40 bits. The source of this delay was the time it took to transmit the R_RDY back to the antenna.

On the surface the R_RDY should have no effect on the end-to-end delay of a frame. Clearly this was not the case and further investigation was required. The key is that as tested the nodes have no inherent delay. This means that when a frame is

received the R_RDY is constructed and sent instantly. The ACK is also constructed and sent instantly. Since the code that is used to construct and send the R_RDY occurs before the code that constructs and sends the ACK, the R_RDY enters the queue first and the ACK must wait until the R_RDY is sent before it can begin transmission. Results of the test are shown in Figure 19 below.

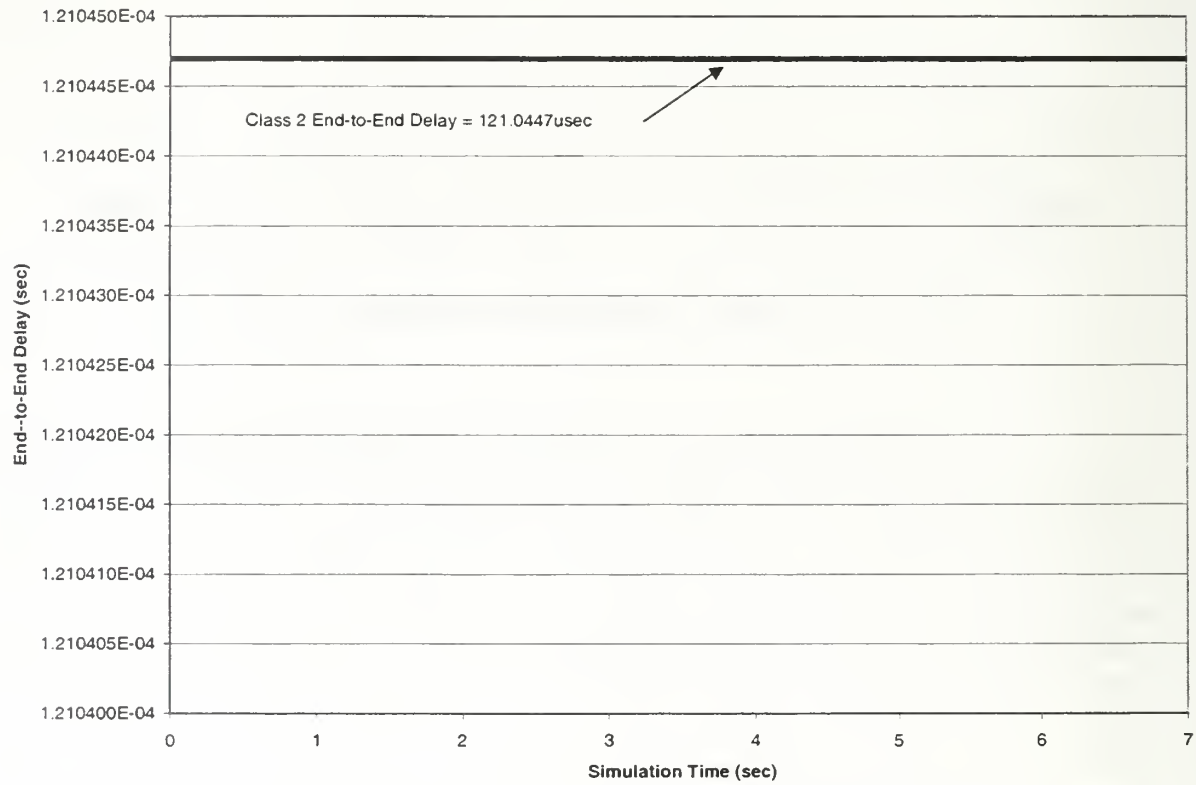

Figure 19. Results of the Class 2 End-to-End Verification Test.

The results of the long-distance test clearly show that the model accurately simulates both the propagation delay and the transmission delays inherent in a Fibre Channel network.

d. Short Distance Model

The short distance model was used to verify that delays through the switches were modeled accurately. The model contained one antenna one processor and two switches connecting the nodes. The model is shown as Figure 20.

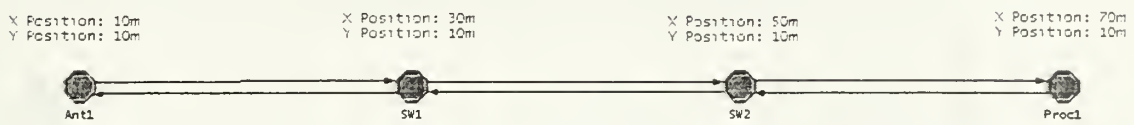

Figure 20. Short Distance End-to-End Test Verification Model

e. Test and Test Conditions

The short distance model was tested using both Class 2 and Class 3 frames. First with Class 2 and then with Class 3. The expected results for Class 3 would be that the delay would be equal to three transmission delays (for the full-length data frames) plus the propagation delay for 60 meters. The approximate value for this would be 61.627usec. For Class 2 one would expect an additional delay of three times the delay for the 600 bit ACK plus the delay for the 40 bit R_RDY plus the propagation delay for the 60-meter return. This equates to an additional 2.032usec or 63.659usec for the round trip.

f. Results

The results of the simulation are contained in Figure 21 below.

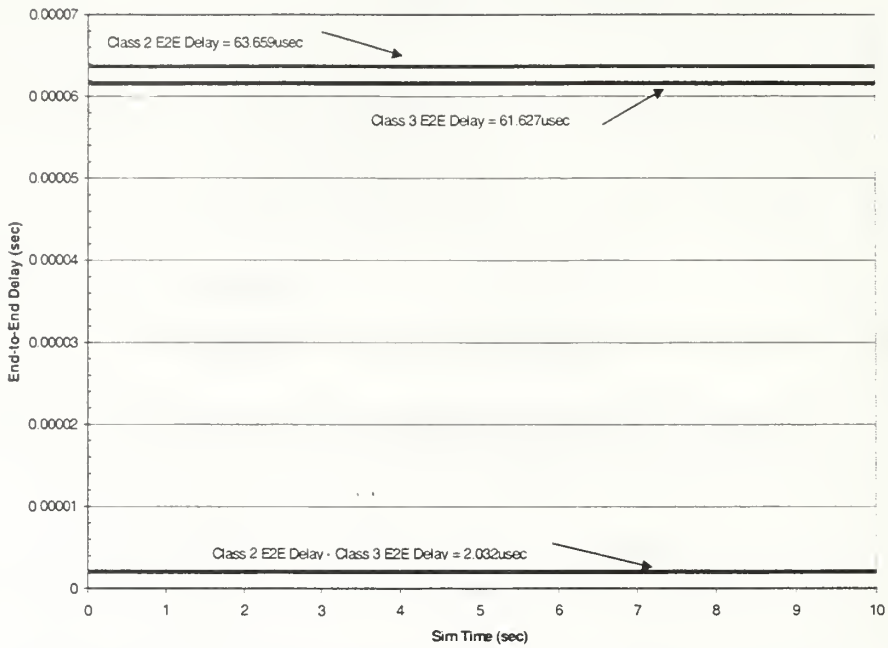

Figure 21. Results of the Short Distance End-to-End Delay Verification Test.

The graph shows that the switch behavior is exactly as predicted, and is suitable for estimating the end-to-end delay inherent in an avionics system.

4. Load Balancing

The last tests that were conducted were used to verify that the bandwidth between switches was shared equally across all links. As was mentioned earlier, the best way to ensure determinism is to have excess bandwidth. By sharing the load between switches one can ensure that all links have equivalent excess bandwidth.

a. Theory

As was stated in the description of the FSM in the node, frames were routed based on a path of least cost. This cost was defined to be the throughput on the link, so that frames would travel down the link with the lowest throughput.

b. Load Balance Model

The model was constructed using one antenna one processor and two switches. Between the two switches four links were connected. The throughput of these links would be monitored to ensure that as simulation time increased the throughput of all the links would be equal. Figure 22 shows the model used for the test.

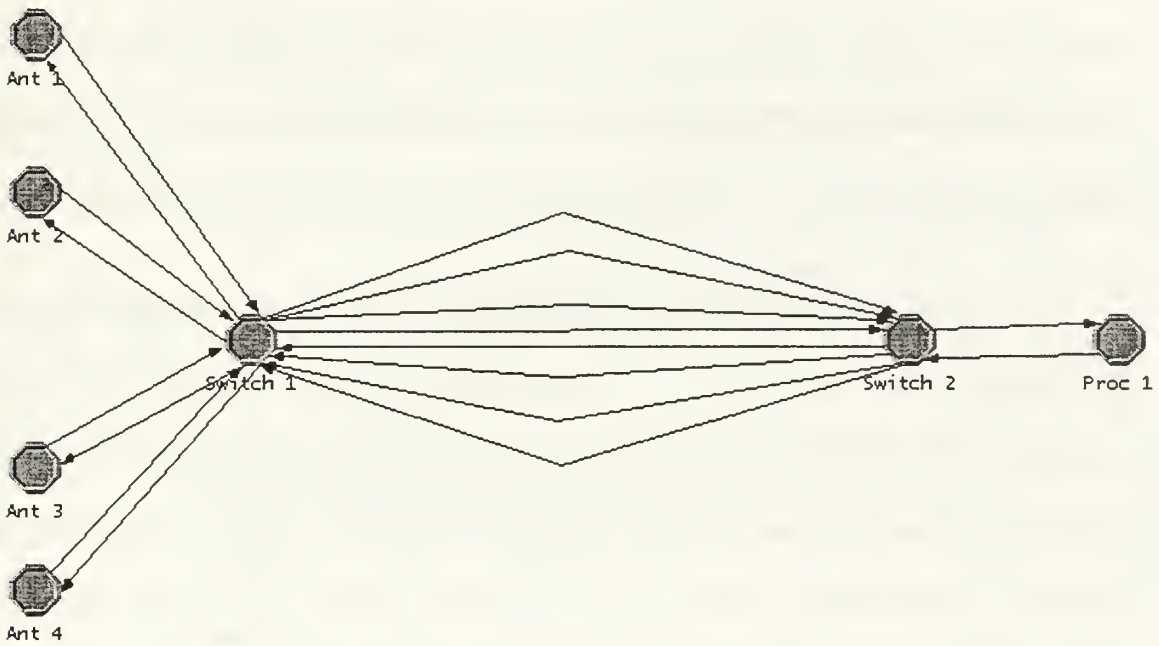

Figure 22. Load Balance Verification Model

c. Test and Test Conditions

The test involved setting the interarrival argument for each of the antennas to 0.004 sec for an aggregate throughput of 21,720,000 BPS. The processor was left in a receive mode only, so that the only data on the links between Switch 1 and Switch 2 would be as a result of the antennas interarrival argument.

The expected result would be that the throughput of each of the four switch links would tend to 5,430,00 BPS and the link between Switch 2 and Proc 1 would be 21,720,000 BPS.

It was assumed that the time it took for the model to reach steady state would be a function of how often the cost of the links was computed. As was mentioned earlier the cost was a function of throughput, and in the switch FSM this cost was calculated applied to the appropriate link on a reoccurring basis. The parameter that controlled this was the Global Variable DEL_COST. Two conditions were tested. The first test used a DEL_COST of 0.5 sec the second test used a DEL_COST of 0.1 sec. Both tests only used Class 3 data since for the path from Switch 1 to Switch 2 using Class 2 data would differ only by the 10,000 BPS required to send the R_RDY frames.

d. ***Results***

Both charts, Figures 23 and 24, show that the load is indeed balanced as the simulation time is increased. This behavior verifies that when multiple links are used to connect switches the load across each of them will tend to become equal. The second result comes from the time required to attain this balanced state. For the first case (Figure 23), where DEL_COST = 0.5 sec the load is balanced ± 5% at approximately 29.00 seconds. For the second case (Figure 24), where DEL_COST = 0.10 sec, the load is balanced ± 5% at approximately 5.48 seconds.

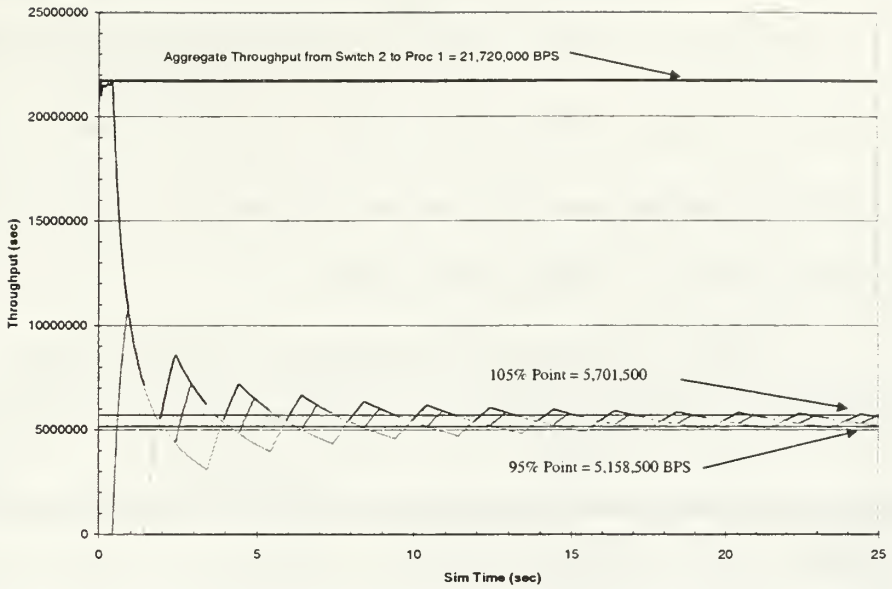

Figure 23. Load Balance Verification Test Result for DEL_COST = 0.5

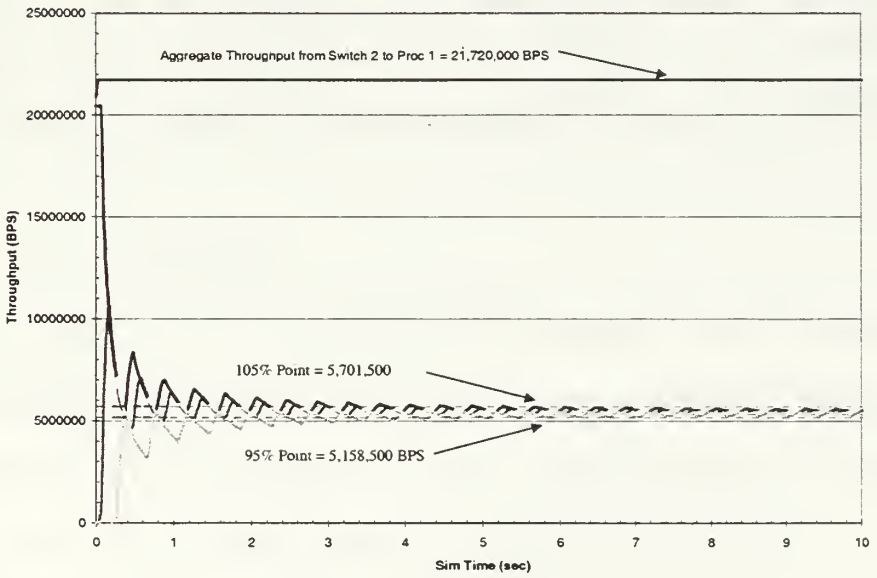

Figure 24. Load Balance Verification Test Result for DEL_COST = 0.10 sec

Upon closer inspection there seemed to be a linear relationship between DEL_COST and the settling time. To determine this a MATLAB function was generated to do a linear least-squares fit of the results of several runs of the load balance test. This

code is contained in Appendix B. Analysis of the data showed that there was a good linear relationship. The equation used for the curve fit was $Settling\_Time = 57.1332 \times DEL\_COST - 0.507$. The data is shown in Table 5. The results of the data and the curve fit are shown in Figure 25. A log-log plot was used to emphasize the points at low values of DEL_COST.

Table 5. Data From the Load Balance Verification Tests

DEL_COST	Measured Settling Time	Calculated Settling Time Using Least Squares Fit	Difference Between Measured and Calculated
0.01 sec	0.53 sec	0.52 sec	0.01 sec
0.05 sec	2.94 sec	2.81 sec	0.13 sec
0.10 sec	5.48 sec	5.66 sec	0.18 sec
0.25 sec	14.24 sec	14.23 sec	0.01 sec
0.50 sec	28.56 sec	28.52 sec	0.04 sec
1.00 sec	57.07 sec	57.08 sec	0.01 sec

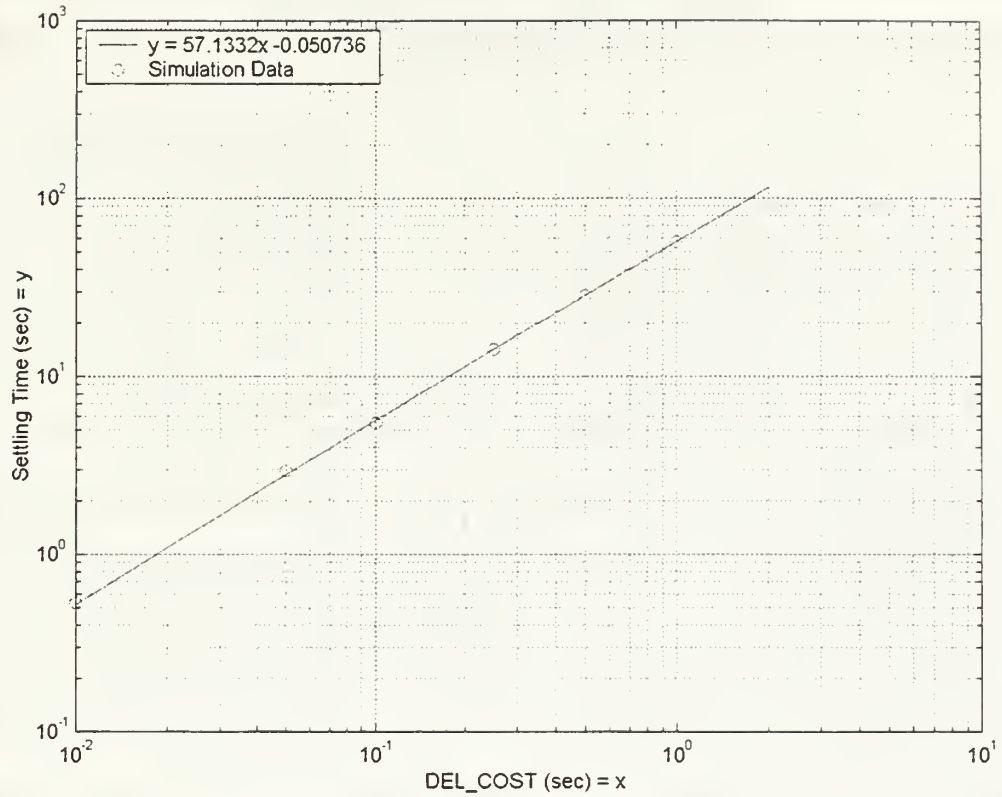

Figure 25. Relationship Between Settling Time and DEL_COST

e. Conclusion

From the perspective of Load Balance the model acts in a predictable way. While some designs may not want to ensure load balance, many will. The utility of the linear relationship between allows the system modeler/designer to predict the consequences of any value of DEL_COST before running the simulation.

C. SUMMARY

As designed the model simulates well many of the characteristics of a Fibre Channel system. The designer/tester of any Fibre Channel system should be aware of the limitations of the model. Future improvements to the model would include modeling of

the Session and Exchange layers. Better modeling of behaviors such as ACK_N, error tolerance etc.

VI. DESIGN, CONSTRUCTION AND TEST OF POSSIBLE AIRCRAFT ARCHITECTURES

A. INTRODUCTION

The purpose for designing this model was first to allow a systems engineer to propose a design for an avionics system and test it under various conditions, and second to take an existing system or component and ensure that it meets the design specifications. During the design phase it was assumed that no components had been specified other than that they had to support Fibre Channel. For the validation phase it was assumed that all components represented actual hardware and could not be changed. This section will attempt to show the design and test of simple systems. Some of the key areas that will be investigated include measuring end-to-end delays, computing appropriate credit values, testing the effect of Class 1 traffic and viewing the effects of system failures. End-to-end delays relate to determinism. Credit requirements relate to the amount of memory required by the different components. Lastly the effects of Class 1 traffic and system failures will show the robustness of the system.

B. DESIGN OF A SIMPLE SYTEM

In order to demonstrate some of the capabilities of the model a simple four node, one switch system was constructed. The system has one radar antenna, one radar processor, one display and one display processor. A single 32-port switch connects these components. The system is designed so that the radar antenna and radar processor can communicate with each other, with the radar processor having the additional capability of talking to the display processor. Only the display processor can communicate with the display. The model is shown in Figure 26.

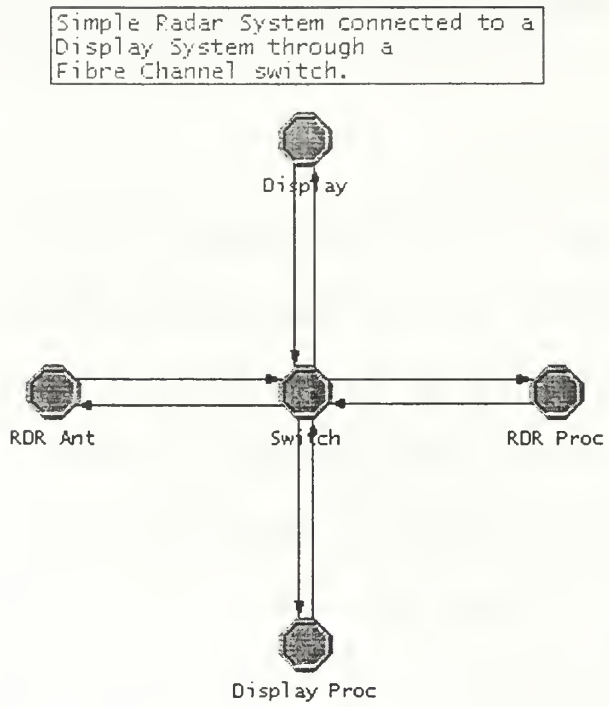

Figure 26. Simple Aircraft Avionics System

1. System Parameters

For this example it was assumed that the desired data throughput from each node was a design point. Table 6 shows the throughputs.

Table 6. Desired Values for Node Throughput

Node	Throughput Out
Antenna	5MBPS
Display	50KBPS
Radar Processor	100KBPS
Display Processor	1MBPS

Since no hardware was specified the system would be designed using several values for delay. It was assumed that the time required to send an R_RDY would be very quick so values of R_RDY Delay were set to 1.0nsec, and 100nsec. The time to forward packets through the switch (Switch attribute, switch.Processing Delay) was defined for 50nsec and 5usec. Processing times for the various nodes (proc.Processing Delay) were

also set to 500nsec and 50usec. It is clear that even for this small network many combinations of delay times exist. For this example one test case was analyzed of the more than 256 possible combinations. The test matrix is shown in Table 8.

Table 7. R_RDY Delay = 100.0nsec, Switch Processing Delay = 500.0nsec

Node	Processing Delay
Antenna	500.0nsec
Display	500.0nsec
Radar Processor	500.0nsec
Display Processor	500.0nsec

The test condition would be run using Class 3 only, Class 2 only and a 50/50 mix of Class 2 and Class 3 messages. The goal of the system designer was to accomplish these throughputs with no lost frames, using as little RAM as possible.

2. Throughput Calculations

As was stated earlier each node only broadcasts full-length data frames, ACK frames and R_RDY primitives. For the case of Class 3, outgoing throughput is approximately a function of interarrival argument. But, since the specification states that the data throughput one must also take into account the effect of 8b/10b encoding, the associated frame overhead, and the number of bits of data to be transmitted for each frame. For each bit of raw data 1.2 bits of Fibre Channel payload are generated. Also, as shown in Table 4, for each frame sent 600 bits of overhead are generated. Taking all this into account, for the condition that full data frames are sent (i.e. 2112 Bytes of data per frame transmitted) throughput on the link must be increased by approximately 28.6%. A MATLAB function was written to calculate interarrival argument based on these parameters, and is contained in Appendix D. In addition, there will be slightly more output due to the transmission of R_RDY primitives in response to Class 3 messages.

For Class 2 messages the output is even greater due to the requirement to transmit one ACK for each Class 2 frame received. Table 9 shows the interarrival arguments calculated which should satisfy the throughput requirements.

Table 8. Calculated Frame Interarrival Arguments

Node	Interarrival Argument (sec/frame)
Antenna	0.003223
Display	0.3300
Radar Processor	0.1650
Display Processor	0.01611

3. Buffer-to-Buffer Credit Calculations

The next step in the design of the system is to ensure that there is enough buffer-to- buffer credit in the nodes as well as in the switch. To do this the MATLAB function user_get_credit.m was written. This code is contained in Appendix B.

a. Node Buffer-to-Buffer Calculations

Node buffer-to-buffer credit required is simply a function of how long it takes a frame to get from the node to the switch. This delay is a function of distance, frame-rate, link throughput, payload size, and number of nodes in the system, and switch delays. For a simple system it is fairly easy to analytically calculate these delays. These factors are fed into the user_get_credit.m function and values for delay are returned. These values can then be verified through simulation. Results from this showed that for small systems with small R_RRDY delays (less than 0.1 sec) buffer-to-buffer credit need not be greater than one. Table 10 shows the delays for the four nodes of interest. These delays are based on the sending of only full payload (21120 bits) Class 3 frames, on a full speed (1,062,500,000BPS) Fibre Channel Link with an assumed propagation speed of 200,000,000 meters/sec and a switch R_RDY delay of 100.0nsec.

Table 9. Time for A Node to receive R_RDY

Node	Distance to Switch	Propagation Delay	Frame Transmission Delay	Switch R_RDY Delay	Total Time to Receive R_RDY
Radar Antenna	10m	50nsec	20.442usesc	37.65nsec	20.68usec
Radar Processor	10m	50nsec	20.442usesc	37.65nsec	20.68usec
Display	10m	50nsec	20.442usesc	37.65nsec	20.68usec
Display Processor	10m	50nsec	20.442usesc	37.65nsec	20.68usec

This chart assumes that the frames experience no other delays, and by the design all delays are constant and therefore completely deterministic. In general one can expect that not only will other delays occur, but also that the delays will have some sort of non-constant distribution. These delays will be small for small systems, but as complexity is increased they cannot be completely discounted.

b. Switch Buffer-to-Buffer Credit Calculations

The calculation required to determine the appropriate amount of credit for the switch is analogous to that required when determining buffer-to-buffer credit to the switch. There is however one minor difference. The switch must have adequate credit so that when more that one node is communicating with another node credit is available. In this example both the Antenna and the Display Processor communicate with the Radar Processor. Over time it can be expected that both will want to communicate simultaneously with the Radar Processor and hence the switch must have at least two buffer-to-buffer credits available for use. This formulation gets increasingly complex as multiple nodes are added and throughput rates are increased. Both factors must be considered in the determination. By allowing the system designer to try multiple values

quickly, this simulation tool will greatly reduce the workload and time required to evaluate/design an avionics system.

4. End-to-End Credit Calculations

For systems using both Class 2 and Class 3 frames the designer must ensure that enough End-to-End credit is available in each node. The steps required to do this are somewhat more abstract that those required for Class 3 only use.

The first step is that the model must be constructed with adequate buffer-to-buffer credit to ensure that no frames are lost. The designer then runs the system for an adequate period of time with all Class 2 frames and evaluates the average end-to-end delay. From this data the user can build a histogram showing the possible delays and the frequency of their occurrence. A histogram for this sample system is shown in Figure 27 below.

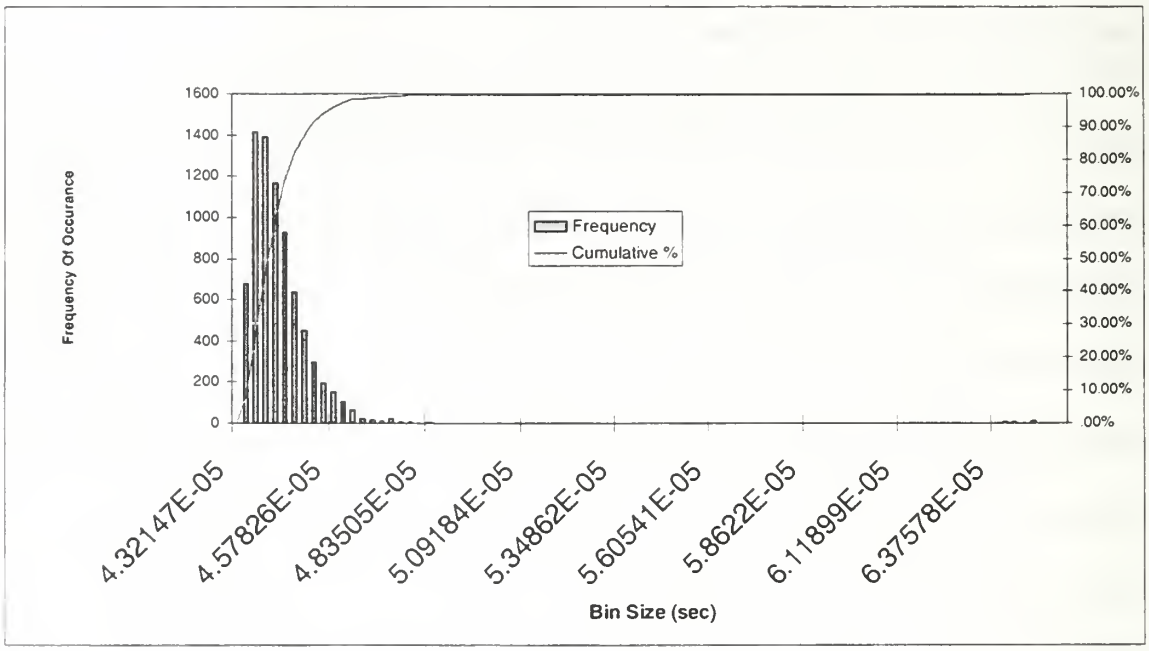

Figure 27. A Histogram Showing the Distribution of End-to-End Delays

It is clear from this data that the delay is never worse than approximately 65usec. In fact upon analysis of the data 99.95% of all delays are less than 65usec. In order to ensure that frames are never lost the worse case delay must be used as one input to the checkCredit.m MATLAB function. The other input is the interarrival argument for the node of interest. For this test the longest delay of the system was 65.56usec. Table 11 shows the end-to-end credit calculations for the system.

Table 10. End-to-End Credit Calculations

NODE	Interarrival Argument	End-to-End Delay	End-to-End Credit Required
Radar Antenna	223.0usec/frame	65.56usec	1
Radar Processor	165.0msec/frame	65.56usec	1
Display Processor	330.0msec/frame	65.56usec	1
Display	16.11msec/frame	65.56usec	1

Two main factors in this network would affect the number of end-to-end credits required. First, if end-to-end delays were to increase by several orders of magnitude each node would require more end-to-end credits to be allocated. For example if the longest delay was greater than 223usec two credits would be required by the Radar Antenna for communications with the Radar Processor. The second way more credits would be needed is by increasing throughput. This increase in throughput would decrease the interarrival argument. If the interarrival argument was decreased below the end-to-end delay value, more credit would be required.

For this simple example the credits required could be calculated by hand. However, as the complexity of the system increases some kind of simulation will be required in order to gain insight into the delays and required credits.

5. System Test

Once all parameters are determined for the system is run to ensure proper behavior. This simple system was run for one minute using a 50/50 mix of Class 2 and Class 3 frames. Full size data payloads were used and throughput was maintained at constant values. Delays for both the switch and all nodes were a constant 500nsec processing delay, and 100nsec R_RDY delay.

a. Buffer-to-Buffer Credit Check

The first data that was examined was to determine if any frames had been lost due to lack of credit at the switch. The statistics that would indicate this are the global statistics "Dead Class 3," "F_BSY LCF," and "F_BSY DAT." All of these statistics registered zero throughout the simulation. This indicated that the switch buffer-to-buffer value of two was sufficient.

The second statistic that would show inadequate buffer-to-buffer credit would be the node statistic "BB Credit Reset." Figure 28 below shows that for the Display, Display Processor and Radar Processor there was insufficient buffer-to-buffer credit for the given loading.

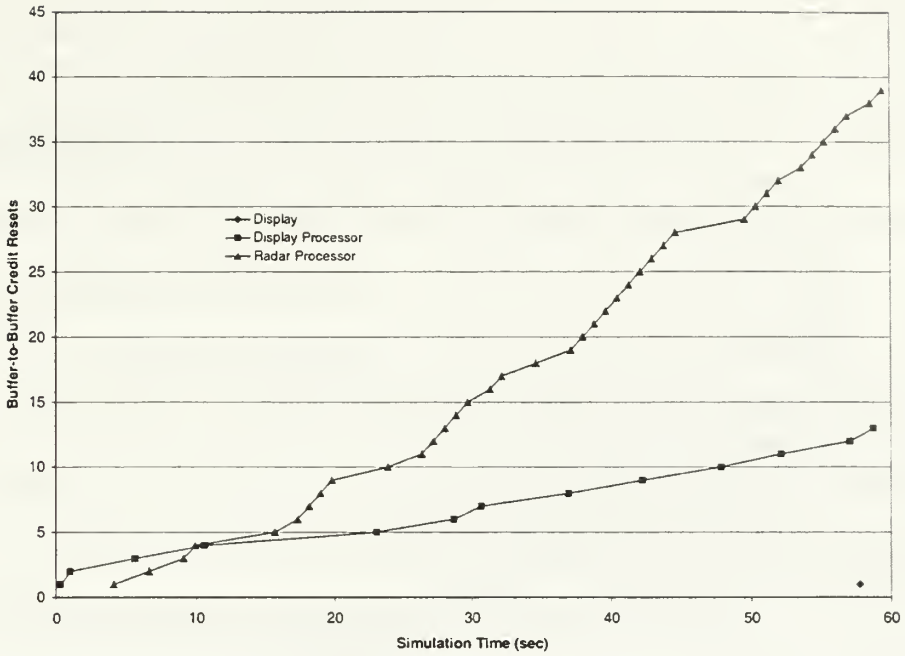

Figure 28. Buffer-to-Buffer Credit Resets For the Full System Test

Clearly the graph shows that there is inadequate buffer-to-buffer credit in the Display, the Display Processor, and the Radar Processor. To correct this deficiency the values for buffer-to-buffer credit were incremented to two and the simulation was re-run. The second run indicated that sufficient buffer-to-buffer credit was available at all the nodes.

b. Throughput

The system's throughput was evaluated using the OPNET tool. Graphs for the four nodes are shown in Figure 29.

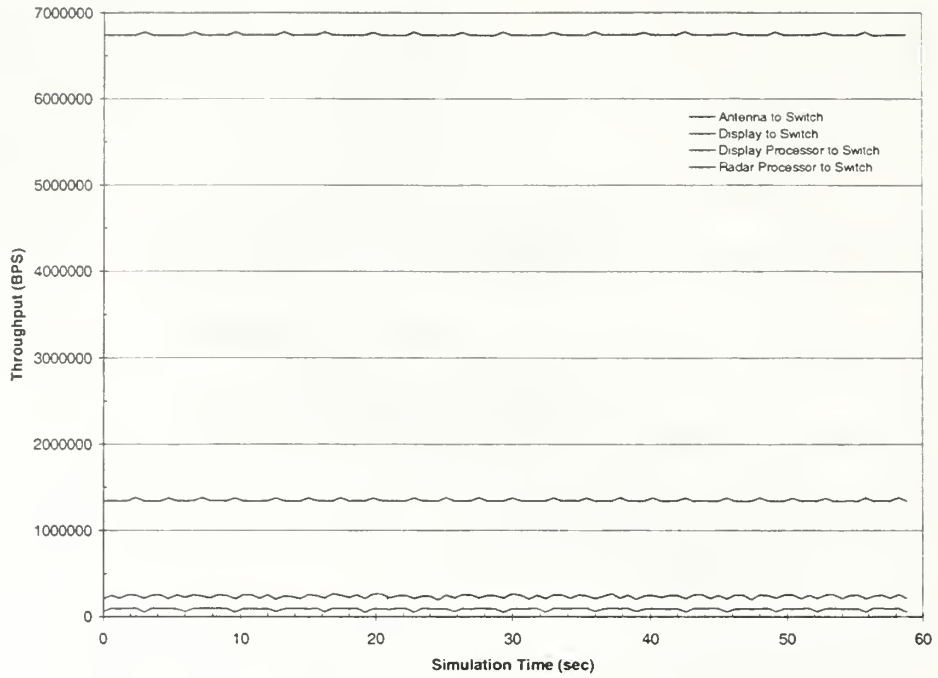

Figure 29. Throughput From All Nodes, 50/50 Mix of Class 2 and 3 Traffic

This graph shows the increase in data throughput due to the use of Fibre Channel as the communications protocol. Table 12 shows the average throughput of the four nodes and the percent increase of throughput due to Fibre Channel Overhead.

Table 11. Increase in Throughput Due to Fibre Channel Protocol

Node	Desired Data Throughput	Actual Throughput	Percent Increase
Radar Antenna	5.0MBPS	6.434MBPS	28.68%
Radar Processor	100.0KBPS	225.39KBPS	125.39%
Display Processor	1.0MBPS	1.288MBPS	28.84%
Display	50.0KBPS	82.247KBPS	64.49%

It is clear that the percent increase is highest for the nodes with the lowest throughput. This is due to the fact that those two nodes are also the recipients of frames from the highest throughput devices, the Radar Antenna and the Display Processor. Since 50% of the frames are Class 2, the node is required to send many ACK frames in

response to the input traffic. This bandwidth utilization passes only administrative traffic with no data payload, and is therefore overhead caused by the selection of Fibre Channel as the communications protocol. For this small system this fact may seem obvious, but the more complex the system the more difficult the traffic pattern is to analyze, and the more valuable a simulation tool becomes.

c. End-to-End Delays

Determinism is critically important in any avionics architecture. The measure of the range of times required for one frame to get from one node to another as well as the time required to receive the ACK that guarantees delivery of a frame is one way to describe how deterministic a system is.

Delays in this model were broken up into Class 3 one way delays and Class 2 end-to-end delays. For the case of Class 3, since there is no guarantee of delivery, the delay was measured from the time the frame was transmitted until it was received successfully at the destination node. This is a one way time delay. For Class 2 the decision was made to measure the time required to receive the acknowledgement that the frame was received from the destination node. This measurement will give the system designer some idea how long he should expect to wait before he can assume that a frame was somehow lost in transmission. In Fibre Channel this value is called the error detect timeout value or E_D_TOV. E_D_TOV is a critical system parameter that can be inferred/verified through use of this simulation tool.

Since all the delays in this system are set as constants there is almost no scatter and it is easy to determine the minimum and maximum delays for the system. Table 13 shows some of the statistics that were gathered from the simulation.

Table 12. End-to-End Statistics

	Max Delay	Min Delay	Percent of Frames at Max Delay	Percent of Frames at Min Delay
Class 2	6.46565E-05	4.42141E-05	0.10%	99.80%
Class 3	6.24271E-05	4.19847E-05	0.20%	99.80%

This data clearly shows that the designer can expect greater than 99% of his frames to arrive at the minimum delay time and less than one percent of the frames to arrive around the maximum delay time. While for this system the histogram does not show any valuable information, in a complex scenario the determinism would not be clear and multiple simulations would have to be run. This OPNET tool gives the designer that capability. Figure 30 below shows the delays in graphical form.

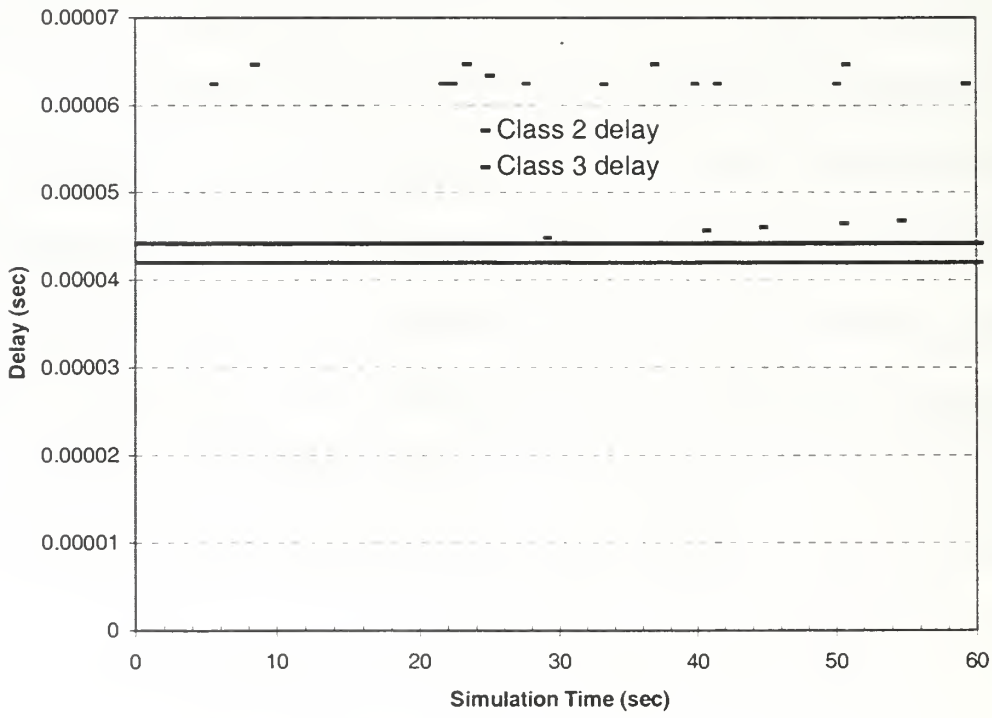

Figure 30. End-to-End Delays for the Example System

VII. CONCLUSION

Simulation is recommended to verify analytical calculations even for a simple, deterministic communications system such as MIL-STD-1553. For a more complex or non-deterministic system, simulation serves as a method of experimenting with new designs prior to implementation. Fibre Channel can be configured in a variety of systems, from simple point to point systems that are completely deterministic, to multi-hop switched fabric systems that are statistical in nature. This thesis has examined Fibre Channel as a potential avionics interconnection standard and described the development of a simulation model that allows detailed simulation of any Fibre Channel system. The model should be of great utility in the design and analysis of systems using the Fibre Channel standard.

A. WHAT THE SIMULATION SHOWED

Even though the simulation did not address many of the nuances of the Fibre Channel Standard it still gave some good insight to the performance one might expect from an avionics system using Fibre Channel as the interconnection system.

1. Determinism and Latency

As the simple simulation showed, even with constant delays at the switches and with only four transmitting nodes, total delays varied from between 42μsec to 64μsec. This performance gets less deterministic as network loading increases and more nodes are placed on the system. Increases in network traffic result in more packets queuing up at each switch or node. This makes it harder to determine exactly when a given packet will arrive at its destination. Clearly, one of the biggest factors in the latency of the packet is how many switches it must transverse. For each switch that is crossed a full size data

packet will pick up 0.94nsec of delay per bit transmitted. This is in addition to any routing delays encountered. For a full size data frame this delay is 20.44μsec.

2. Problems With Displays

One of the areas of concern when using Fibre Channel is the amount of bandwidth required to support high-resolution displays. To illustrate this point assume that the display is a 10 inch by 10-inch display with a resolution of 1024 pixels by 1024 pixels. This equates to 1,048,576 pixels per display. If 24 bits per pixel are used and the screen is refreshed at a 30 Hz rate, the resultant bandwidth is 720Mbits/sec. If Class 3 data frames are used to send this traffic and no other traffic is on this link, the required throughput is approximately 926Mbits/sec. This represents a bandwidth utilization of over 90% on a full speed Fibre Channel link.

Clearly, to support modern day displays one of three solutions needs to be imposed. First, some of the processing burden of the node of interest could be shifted to the display itself. This is the concept of the "Smart Display." Second, some sort of data compression could be used. Lastly, a dual speed (2 Gbits/sec) or quad speed (4 Gbits/sec) could be used for displays and their associated processors.

3. Burden of Administrative Overhead

Systems with increasing complexity and an increasing need for determinism will need additional administrative packets active on the network. These packets will reduce the amount of bandwidth available for actual data. Examples of this sort of traffic include arbitration packets that will determine which packets have priority in a network as well as routing packets that determine optimal routing between various nodes. Also, the choice of transmission class can cause differing quantities of acknowledgement packets to be returned. If the return channel is already heavily loaded, too many

acknowledgement packets may overload the system. As these packets are returned over a different channel than the data they are acknowledging, it is easy to overlook their impact on the system. Clearly, fine grain analysis needs to be done to determine the effect of this overhead on overall system performance and bandwidth utilization. Stating this another way, one cannot assume that just because a node only has an output of 600Mbits/sec that it can be inserted in a 1 Gbits/sec network and expect it to have no impact on overall system performance.

4. Difficulty In Analytical Estimates For Complex Systems

Even in the small system built for this thesis, analytical analysis was difficult. When the network size is increased, the delays become not only less deterministic, but also functions of multiple variables. This will ensure that accurate analytical analysis will be impossible.

B. WHY FIBRE CHANNEL CAN WORK

Fibre Channel is gaining momentum in the commercial industry. Large amounts of research are being conducted to improve the standard. As this continues it is increasingly likely that more and more applications will be found for its use, and cost will continue to decrease. There is an active working group (FC-AE) addressing the specific needs of avionics architectures. Currently, it is being used in such programs as the F-18, B-1B and AWACS aircraft [Ref. 14]. The lessons learned from these implementations will help ensure success of future programs.

Fibre Channel meets all the technical requirements of an avionics interconnect. Specifically, it is a deterministic, real time serial bus capable of sending many types of data between microprocessors. It is scalable, plug-and-play, hot swappable, and easy to maintain. Table 14 shows how well the Fibre Channel switched fabric topology meets

the goals of the JAST Mission Systems IPT. Further details concerning the use of Fibre Channel in an avionics system are contained in Chapter II of this thesis.

Table 13. Digital Network Requirements. After Ref. [1].

Requirement	Third Generation	Fourth Generation	Fibre Channel Switched Fabric
Network Size	<32 connects	<256 connects	Exceeds
Data Rate-per path	400 Mb/s	2.5 Gbps	Fails at 1 Gbps, Exceeds at Quad Speed 4 Gbps
Data Rate-aggregate	10+ Gbps	50-100 Gbps	Meets
Data Integrity-streaming	10E-7 BER	10E-7 BER	Exceeds
Data Integrity-packets	10E-10 BER	10E-10 BER	Exceeds
Packet size	Unlimited	Unlimited	Fails, 2K Bytes
Path Latency (μsec)	<100	<10	Implementation Dependent
Link Control	Self-Routing	Self-Routing	Meets
Blocking	Limited	Non-Blocking	Implementation Dependent
Packaging	SEM-E	SEM-E	Implementation Dependent

There appears to be no technical reasons that Fibre Channel could not replace the current avionics backbone: MIL-STD-1553B.

C. WHY FIBRE CHANNEL MAY NOT WORK

The majority of the work in Fibre Channel has been for storage area networks. At a recent three-day industry exposition (FCTC 99, San Jose, CA) of Fibre Channel products and presentations, there were no companies noted that were currently working on any military applications.

The standard is weak and has loopholes. There have been many instances of two devices that conform to the standard, and yet when connected together the system will not work as a whole. In his keynote address at FCTC 99, Richard Lary explains that, due to the many possible interpretations of the standard, it is likely that any two devices from

different companies will not work together. This fact will severely hinder the design of truly open systems architectures.

Lastly, while the interface to a fabric switch conforms to a standard, the insides of the switch are proprietary. This may make it difficult to second source the switch, further limiting the choices for other commercial alternatives.

D. FOLLOW ON WORK

The following paragraphs address the shortcomings of this model.

1. Session and Exchange Layers

This thesis only models the physical and link layers of Fibre Channel. Important additional insight to Fibre Channel's performance could be obtained if the model's fidelity was increased to include the session and exchange layers. The addition of these layers in the simulation would quantify the expected increased bandwidth utilization and decreased determinism that would come as a result of the use of sessions and exchanges.

2. Probability Density Function Models of Real Systems

During the testing and simulation of the simple avionics architectures there was no real reason for the choice of constant delays. In reality the delays would follow some sort of PDF. This PDF would have to be obtained using either vendor data or a Monte Carlo simulation based on some assumptions about the switches and various nodes. The subsequent use of this PDF would add fidelity to the model.

3. Comparison of Simulation Data to Real System

Ultimately, the quality of a simulation is determined when the results of the simulation compare well to actual data obtained from system tests. Unfortunately, at the time of this writing no data was available for comparison. Future research in this area should try to obtain throughput, overhead and end-to-end delay data from an actual system and compare that data to data generated by the simulation.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. COMPARISONS OF OTHER INTERCONNECT SYSTEMS

1. OVERVIEW

This purpose of this thesis is to show how well Fibre Channel suits the next generation avionics interconnect. The following are some of the competing technologies and why they may or may not be a suitable alternative.

a. Firewire

Firewire or IEEE-1394 as it is also known is a fast (100Mbit/sec to 400Mbit/sec), peer-to-peer serial bus. It supports hot swapping of devices and can be used to transport a large variety of data. It is working to become the standard interface of the desktop computer.

Firewire has several limitations. First, the network is in the form of a tree topology with limited distance between devices. Second, since it's market appears to be limited to the desktop market, very few ruggedized components are commercially available. [Ref. 14] It is unlikely that IEEE 1394 will gain a large share of the military interconnect market, and hence a somewhat risky technology upon which to base future avionics architectures.

b. Gigabit Ethernet

Since it lives in the collision domain, Ethernet is inherently non-deterministic. This fact alone makes it unsuitable for an avionics application.

c. Asynchronous Transfer Mode

Asynchronous Transfer Mode (ATM) is a fast (622 Mbits/sec) [Ref. 15] serial bus designed for the telecommunications industry. Primarily designed for voice/video, it's disadvantages include that it is a lossy protocol, has high software overhead and that it has had many delays in both hardware and software standardization. [Ref. 14] (It is interesting to note that ATM protocol can be run over a Fibre Channel network.) Also, there does not appear to be a preponderance of ruggedized components for ATM, which are necessary for use in a military combat aircraft. ATM does not appear suitable for use in avionics architectures at this time.

APPENDIX B. OPNET NODE FINITE STATE MACHINE CODE

1. LIST OF STATE VARIABLES

Table B-1. List of State Variables Used in the Node Finite State Machine

Variable Name	Variable Type	States where variable is used
s_id	int	objid?, FC_1, b_plate, route, snd_rrdy
subnet	int	subnet?, b_plate, route
d_id_list[200]	Objid	get_pkt, FC_0, FC_1, others, computers
subnet_list[200]	Objid	get_pkt, route, others, computers
num_proc	int	get_pkt
class_dist	Distribution*	init
buf_to_buf	int	init, xmt_pkt, b_to_b, check_credit, r_rdy_rec
rte_arry[200]	Route*	route, topo_build
ete_gsh	Stathandle	init, FC_0
ete_gsh_cl3	Stathandle	init, FC_1
f_bsy_dat_gsh	Stathandle	init, FC_0
f_bsy_lcf_gsh	Stathandle	init, FC_0
proc_delay	double	objid?, xmt_pkt
r_rdy_delay	double	objid?, snd_rrdy
max_buf	int	init, b_to_b, check_credit, r_rdy_rec
e_to_e[200]	int	computers, e_to_e, others
no_bb_count	int	init, b_to_b, check_credit
no_bb_count_sh	Stathandle	init, B_to_b, check_credit
no_ee_count	int	init, e_to_e
no_ee_count_sh	Stathandle	init, e_to_e
login_e_to_e[200]	int	computers, others, e_to_e

2. LIST OF TEMPORARY VARIABLES

Table B-2. List of Temporary Variables Used in the Node Finite State Machine

Variable Name	Variable Type	States where variable is used
node_id	Objid	objid?, b_plate
node_type[10]	char*	objid?, function?
node_name[10]	char*	objid?
subSystem[10]	char*	objid?, computers, others
subnet_id	Objid	subnet
to_whom	int	function?, i talk to
what_subsys	int	function?, computers, others
to_computers	Boolean	i talk to
to_others	Boolean	i talk to
total_nodes	int	computers, others
temp_ids[200]	Objid	computers, others
tmp_char[10]	char*	computers, others
pkptr	Packet*	get_pkt, rcv, CL1_xmt, CL2_xmt, CL3_xmt, FC_0, FC_1, xmt_pkt, b_plate, snd_rrdy, r_rdy_rec
d_id	int	get_pkt, FC_1, b_plate, route
dst_subnet	int	get_pkt, FC_1, route
dest_index	int	get_pkt, route, FC_1, e_to_e
class_type	int	b_plate
data_size	int	b_plate
rset_ptr	Route_Set*	route
tmp_rte	Route*	route
numFields	int	rcv
c3Rrdy	int	rcv
fc_type	int	snd_rrdy
src_nde	int*	snd_rrdy
rdyptr	Packet*	snd_rrdy
classType	int	FC_1

class_3_rec	int	FC_1
create_time	double	FC_1, FC_0
ete_delay3	double	FC_1
ete_delay	double	FC_0
f_bsy	int	FC_0
src_subnet	int	FC_0
src_id	int	FC_0

3. HEADER BLOCK

The following code is contained in the header block of the node's process model.

```
/*****************************************************************/

/* packet stream definitions */
#define RCV_IN_STRM 0       //Packet which comes from the node's receiver
#define SRC_IN_STRM 1       //Packet which comes from the ideal source
#define XMT_OUT_STRM 0 //Packet which is sent to the transmitter

/* transition macros are used to transition between states*/
#define SRC_ARRVL1 ( op_intrpt_type () == \
        OPC_INTRPT_STRM && op_intrpt_strm () == SRC_IN_STRM )

#define RCV_ARRVL1 ( op_intrpt_type () == \
        OPC_INTRPT_STRM && op_intrpt_strm () == RCV_IN_STRM )

#define REROUTE (op_intrpt_type() == OPC_INTRPT_SELF)

#define COMPUTERS ( to_computers == OPC_TRUE )

#define OTHERS ( to_others == OPC_TRUE )
```

```
#define CLASS_1_XMT (class_type == 1)
#define CLASS_2_XMT (class_type == 2)

#define CLASS_3_XMT (class_type == 3)

#define FC_0_RCV (fc_type == 0xC)                    //Link Control Frame Recvd

#define FC_1_RCV (fc_type >= 0x2 && fc_type < 0xC)   //Data Frame Recvd

#define CLASS_3_REC (class_3_rec == 1)

#define OTHER_CLASS_REC (class_3_rec == 0)

#define R_RDY_RCV (c3Rrdy == 1)

#define FRAME_REC (c3Rrdy == 0)                      //Either LCF or Data


//*********************************************************************//
//                          Global Variables                           //
//                                                                     //
//*********************************************************************//
const int MAX_DATA_SIZE = 21120;      //Maximum Data Bits allowed in one Frame
const int MIN_DATA_SIZE = 0;          //Define MIN_DATA_SIZE to 0 bits
Topology     *TOPO_PTR;               //The only Topology Pointer
const  int   MAX_HOPS = 5;            //Max number of Hops a path can have
double       LAST_TOPO_UPDATE_TIME = -1;
double       FIRST_AWAKE = -1;  // Make sure Statistics are only initialized once.
const  double      DEL_T = .5;              // Time between topology builds
int                F_BSY_DAT;               // Holds the F_BSY_DAT stats
int                F_BSY_LCF;               // Holds the F_BSY_LCF stats
```

```
//*************************************************************************//
//                          Function Prototypes                            //
//                                                                         //
//*************************************************************************//
// Function: int data_sz_set(int)
// Purpose:  Returns data with size 0 - 21120 bits

        int data_sz_set(int);

//end Header
/*************************************************************************/
```

4. FUNCTION BLOCK

The following code is contained in the function block.

```
/*************************************************************************/
// Function: int data_sz_set(int)
// Purpose:  Returns data with size 0 - 21120 bits
int data_sz_set(int dta_size)  {
    int max_data_size;
//max_data_size gives an exponential  dist of data with mean of dta_size
//dta_size in in words of data payload, while max_data_size is in bits
    max_data_size = 40*(int)op_dist_exponential(dta_size);
 if (max_data_size >= MAX_DATA_SIZE)   {      // i.e greater than 21120 bits
                        return MAX_DATA_SIZE;
                }   else if (max_data_size < MIN_DATA_SIZE)   {
                        return MIN_DATA_SIZE;
                }       else    {
                        return max_data_size;
                } //end if
        }//  end data_sz_set
/*************************************************************************/
```

5. INIT STATE

The following is the code from the enter executives of the **init** state in the Fibre Channel node FSM.

```
/*****************************************************************/
//declare global statistics (_gsh indicates a global statistic handle)
ete_gsh = op_stat_reg ("ETE Delay",
        OPC_STAT_INDEX_NONE, OPC_STAT_GLOBAL );  //for all packets

ete_gsh_cl3 = op_stat_reg ("CL 3 ETE Delay",
        OPC_STAT_INDEX_NONE, OPC_STAT_GLOBAL ); //only for class 3

f_bsy_dat_gsh = op_stat_reg ("F_BSY DAT",
        OPC_STAT_INDEX_NONE, OPC_STAT_GLOBAL );

f_bsy_lcf_gsh = op_stat_reg ("F_BSY LCF",
        OPC_STAT_INDEX_NONE, OPC_STAT_GLOBAL );

//init the state variables for counting credit resets
no_bb_count = 0;
no_ee_count = 0;

//register the local statistics
no_bb_count_sh = op_stat_reg("BB Credit Reset",
        OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL);

no_ee_count_sh = op_stat_reg("EE Credit Reset",
        OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL);


// if you're the first one awake initialize the statistics variables.
```

```
if (FIRST_AWAKE < op_sim_time () ) {
        FIRST_AWAKE = op_sim_time ();
        F_BSY_DAT = 0;
        F_BSY_LCF = 0;
        }

// load the class distribution (class_pdf is defined in the PDF editor)
class_dist = op_dist_load ("class_pdf", 0.0, 0.0);

//ititialize the buf_to_buf credit
op_ima_obj_attr_get ( op_id_self (), "Buf to Buf", &buf_to_buf);

max_buf = buf_to_buf;  //can't go higher than this, equivalent to login credit

//schedule the first topology building interrupt
op_intrpt_schedule_self ( op_sim_time (), 0);

//*********************************************************************//
//                          Global Variables                           //
//                                                                     //
//*********************************************************************//
        //int       F_BSY_DAT:      Holds data for F_BSY's to data frames.
        //int       F_BSY_LCF:      Holds data for F_BSY's to LCF frames.
        //double    FIRST_AWAKE:    Time the first node wakes up.

//*********************************************************************//
//                          State Variables                            //
//                                                                     //
//*********************************************************************//
//Distribution* class_dist:     Holds the dist of the class of packets sent
//Stathandle    ete_gsh:        SH to the ete global stat.
```

```
//Stathandle    ete_gsh_cl3:    SH to the ete_cl3 global stat.
//Stathandle    f_bsy_dat_gsh:  SH to the f_bsy to data stat.
//Stathandle    f_bsy_lcf_gsh:  SH to the f_bsy to LCF stat.
//Stathandle    no_bb_count_sh: SH to the bb_count stat.
//Stathandle    no_ee_count_sh: SH to the ee_count stat.
//int           no_bb_count:    Count of the number of times b_to_b is reset
//int           no_ee_count:    Count of the number of times e_to_e is reset
//int           buf_to_buf:     Buf_to_buf credits.
//int           max_buf:        Login value of buf_to_buf credits

//*************************************************************************//
//                          Temporary Variables                            //
//                                                                         //
//*************************************************************************//
//                               NONE

//end init
/*************************************************************************/
```

6. OBJID? STATE

The following code is from the **objid?** state in the Fibre Channel node FSM.

```
/*************************************************************************/
// This module finds out who the node is (name, type) and what
//subsystem it is part of.  Also the State variables proc_delay and
//r_rdy_delay are set.

node_id = op_topo_parent(op_id_self());
s_id = node_id;

//get attributes of the node model
```

```
op_ima_obj_attr_get (node_id, "model" , &node_type);
op_ima_obj_attr_get (node_id, "name"  , &node_name);

//get attributes of the process model
op_ima_obj_attr_get (op_id_self (), "Subsystem", &subSystem);
op_ima_obj_attr_get (op_id_self (), "Process Delay", &proc_delay);
op_ima_obj_attr_get (op_id_self (), "R_RDY Delay", &r_rdy_delay);

/**************************************************************************/
//                              Global Variables                          //
/**************************************************************************/
//                                 NONE

/**************************************************************************/
//                              State Variables                           //
/**************************************************************************/
//int           s_id:   The source id of this node
//double    proc_delay:   The processing delay associated with this node
//double    r_rdy_delay: The delay assoc with sending an r_rdy

/**************************************************************************/
//                              Temporary Variables                       //
/**************************************************************************/
//Objid       node_id:          The Objid of this node.
//char*       node_type[10]:    The type of node (i.e. proc, disp, ant)
//char*       node_name[10]:  The name of the node.
//char*       subSystem[10]:  The subsystem of the node (EW, NAV, RDR etc.).

//end objid?
/**************************************************************************/
```

7. SUBNET? STATE

The code for the **subnet?** state is contained below.

```
subnet_id = op_topo_parent(node_id);
subnet = subnet_id;
//printf("My subnet # is: %d \n", subnet);

/*****************************************************************/
//                          Global Variables                     //
/*****************************************************************/
//                             NONE
/*****************************************************************/
//                          State Variables                      //
/*****************************************************************/
//int          subnet:       The subnet of this node
/*****************************************************************/
//                          Temporary Variables                  //
/*****************************************************************/
//Objid    subnet_id:   The Objid of the subnet of this node.

//end subnet?
/*****************************************************************/
```

8. FUNCTION? STATE

The following code is contained in the **function?** state in the Fibre Channel node FSM.

```
/*****************************************************************/
// This module finds out the function of the node inside the system.
//This information will help build the list of nodes that this node
//will talk to.  Each node has a type and a subSystem.
```

```
if (strcmp(node_type,"FC_ant")==0)  {
        //printf("I am an antenna\n");
        to_whom = 1;

}       else if (strcmp(node_type,"FC_proc")==0)   {
                printf("I am a computer\n");
                to_whom = 2;

        }       else if (strcmp(node_type,"FC_disp")==0)   {
                        //printf("I am a display\n");
                        to_whom = 3;

                }       else    {
                                //printf("I am not working\n");
                                to_whom = 0;

                        } //end if

if (strcmp(subSystem,"EW")==0)  {
        printf("I am in EW ss.\n");
        what_subsys = 1;

}       else if (strcmp(subSystem,"EO")==0)         {
                printf("I am in EO ss.\n");
                what_subsys = 2;

        }       else if (strcmp(subSystem,"NAV")==0)      {
                        printf("I am in Nav ss.\n");
                        what_subsys = 3;
```

```
}       else if  (strcmp(subSystem,"RDR")==0)      {
                printf("I am in RDR ss.\n");
                what_subsys = 4;

        }       else if (strcmp(subSystem,"DISP")==0)       {
                        printf("I am in Disp ss.\n");
                        what_subsys = 5;

                }//end else if
/*****************************************************************/
//                          Global Variables                     //
/*****************************************************************/
//                              NONE
/*****************************************************************/
//                          State Variables                      //
/*****************************************************************/
//                              NONE
/*****************************************************************/
//                          Temporary Variables                  //
/*****************************************************************/
//char*     node_type[10]:  The type of node (i.e. proc, disp, ant)
//char*     subSystem[10]:  The subsystem of the node (EW, NAV, RDR etc.).
//int       to_whom:        Used to specify who the node talks to
//int       what_subsys:    What subsytem am I a part of

//end function?
/*****************************************************************/
```

9. I TALK TO STATE

The following code is contained in the **i talk to** state that is in the Fibre Channel node FSM.

```
/*****************************************************************/
// This module aids the FSM in its transition to the building of the communications
//lists.

        switch (to_whom)      {

                case 0: {
                        //printf("I talk to no-one\n");
                        break;
                }

                case 1: {
                        //printf("I talk to computers\n");
                        to_computers = OPC_TRUE;
                        break;
                }

                case 2: {
                        //printf("I talk to displays, computers and antennas\n");
                        to_others = OPC_TRUE;
                        break;
                }
                case 3: {
                        //printf("I talk to display computers\n");
                        to_computers = OPC_TRUE;
                        break;
                }

        } //end switch
```

```
/*****************************************************************/
//                          Global Variables                     //
/*****************************************************************/
//                              NONE
/*****************************************************************/
//                          State Variables                      //
/*****************************************************************/
//                              NONE
/*****************************************************************/
//                          Temporary Variables                  //
/*****************************************************************/
//int           to_whom:      Used to specify who the node talks to
//Boolean       to_computers: True/False:  I talk to computers?


//end i talk to
/*****************************************************************/
```

10. COMPUTERS STATE

The following code is contained in the **computers** state in the Fibre Channel node FSM.

```
/*****************************************************************/
//If you talk to computers the computers state will build arrays of the objects in the
//simulation that you will talk to.
/*****************************************************************/
// count the nodes
total_nodes = op_topo_object_count (OPC_OBJMTYPE_NODE);
/*****************************************/
// initalize the arrays, –99 is a bogus Objid

for (i = 0; i < 200; i++)        {
        subnet_list[i]  = -99;
```

```
        d_id_list[i]    = -99;
        e_to_e[i]               = -99;
}
/*****************************************/
//printf("the total number of nodes in the system is %d \n",total_nodes);
/*************************************************************************/
//  This loads all the nodes into a temporary array

for (i = 0; i< total_nodes; i++)        {
        temp_ids[i] = op_topo_object(OPC_OBJMTYPE_NODE, i);
}  // end for

/*************************************************************************/
switch (what_subsys) {
        case 1: {               //EW subsys
                j=0;    //initialize the counter j

                for (i = 0; i< total_nodes; i++)        {
                        //get the model name
                        op_ima_obj_attr_get   (temp_ids[i], "model", &tmp_char);

                        //if it's a computer
                        if ( strcmp (tmp_char,"FC_proc" ) == 0 )    {
                          op_ima_obj_attr_get (op_topo_child(temp_ids[i],
                                OPC_OBJTYPE_PROC, 0),   "Subsystem", &subSystem);


                           //and it's an EW computer add it to the list
                 if ( strcmp (subSystem, "EW") == 0 )       {
                  op_ima_obj_attr_get(op_topo_child(temp_ids[i],
                  OPC_OBJTYPE_PROC,  0), "End to End", &login_e_to_e[j]);
                        e_to_e[j] = login_e_to_e[j];
                        subnet_list[j]  = op_topo_parent(temp_ids[i]);  //load the subnet
                        d_id_list[j]    = temp_ids[i];                  //load the did
```

```
                j++;                                    //increment j
                        }
                    } //end if
                } //end for
        break;
}//end case 1

case 2: {               //EO subsys
        j=0;    //initialize the counter j
        for (i = 0; i< total_nodes; i++)        {

        //get the model name
        op_ima_obj_attr_get  (temp_ids[i], "model", &tmp_char);

                if ( strcmp (tmp_char,"FC_proc" ) == 0 )    { //if it's a computer
                op_ima_obj_attr_get  (op_topo_child(temp_ids[i],
                OPC_OBJTYPE_PROC, 0),  "Subsystem", &subSystem);

                        //and it's an EO computer add it to the list
                        if ( strcmp (subSystem, "EO") == 0 ) {
                op_ima_obj_attr_get(op_topo_child(temp_ids[i],
                OPC_OBJTYPE_PROC, 0), "End to End", &login_e_to_e[j]);
                e_to_e[j] = login_e_to_e[j];
                subnet_list[j]  = op_topo_parent(temp_ids[i]);  //load the subnet
                d_id_list[j]    = temp_ids[i];                  //load the did
                j++;                                    //increment j
                        } //end if
                    } //end if
                } //end for
        break;
}//end case 2
```

```
case 3: {                   //NAV
        j=0;    //initialize the counter j
        for (i = 0; i< total_nodes; i++)        {

//get the model name
        op_ima_obj_attr_get  (temp_ids[i], "model", &tmp_char);
        if ( strcmp (tmp_char,"FC_proc" ) == 0 )    {       //if it's a computer
                op_ima_obj_attr_get  (op_topo_child(temp_ids[i],
                OPC_OBJTYPE_PROC, 0),   "Subsystem", &subSystem);

          //and it's a NAV computer add it to the list
          if ( strcmp (subSystem, "NAV") == 0 )     {
            op_ima_obj_attr_get(op_topo_child(temp_ids[i],
              OPC_OBJTYPE_PROC, 0), "End to End", &login_e_to_e[j]);
            e_to_e[j] = login_e_to_e[j];
            subnet_list[j]      = op_topo_parent(temp_ids[i]);  //load the subnet
            d_id_list[j]        = temp_ids[i];                  //load the did
            j++;
                                }
                        }  //end if
                }   //end for
        break;
}//end case 3

case 4: {                   //RDR subsys
        j=0;    //initialize the counter j

        for (i = 0; i< total_nodes; i++)        {
        //get the model name
          op_ima_obj_attr_get(temp_ids[i], "model", &tmp_char);
            if ( strcmp (tmp_char,"FC_proc" ) == 0 )        {  //if it's a computer
```

```
                op_ima_obj_attr_get  (op_topo_child(temp_ids[i],
                  OPC_OBJTYPE_PROC, 0),  "Subsystem", &subSystem);

                //and it's a RDR computer add it to the list
                if ( strcmp (subSystem, "RDR") == 0 )        {
                  op_ima_obj_attr_get(op_topo_child (temp_ids[i],
                    OPC_OBJTYPE_PROC, 0), "End to End", &login_e_to_e[j]);
                  e_to_e[j] = login_e_to_e[j];
                  subnet_list[j] = op_topo_parent(temp_ids[i]); //load the subnet
                  d_id_list[j]   = temp_ids[i];                 //load the did
                  j++;
                              }
                        }  //end if
                }   //end for
        break;
}//end case 4

case 5: {               //DISP subsys
        j=0;    //initialize the counter j

        for (i = 0; i< total_nodes; i++)          {
        //get the model name
          op_ima_obj_attr_get(temp_ids[i], "model", &tmp_char);
            if ( strcmp (tmp_char,"FC_proc" ) == 0 )          {  //if it's a computer


                op_ima_obj_attr_get  (op_topo_child(temp_ids[i],
                  OPC_OBJTYPE_PROC, 0),  "Subsystem", &subSystem);

                //and it's a DISP computer add it to the list
                if ( strcmp (subSystem, "DISP") == 0 )        {
```

```
                    op_ima_obj_attr_get(op_topo_child (temp_ids[i],
                      OPC_OBJTYPE_PROC, 0), "End to End", &login_e_to_e[j]);
                    e_to_e[j] = login_e_to_e[j];
                    subnet_list[j] = op_topo_parent(temp_ids[i]); //load the subnet
                    d_id_list[j]   = temp_ids[i];                 //load the did
                    j++;
                                }
                          } //end if
                    }  //end for
            break;
        }//end case 5
}//end switch
/*************************************************************************/
j=0;    //initialize j
/*************************************************************************/
//This just prints out the list of processers in (Sub_net id, d_id) form

while (op_topo_parent(d_id_list[j]) != OPC_OBJID_INVALID)    {
        printf("(%d , %d) \n",subnet_list[j], d_id_list[j]);
        j++;
}               //end while

num_proc = j;
printf("Computers J is equal to %d \n",j);
//end computers enter execs


/*************************************************************************/
//                              Global Variables                         //
/*************************************************************************/
//                                  NONE
```

```
/*****************************************************************/
//                          State Variables                      //
/*****************************************************************/
//int   d_id_list[200]:      Destinations I might talk to
//int   subnet_list[200]:    Their assoc. subnets
//int   e_to_e[200]:         End_to_end credit of the connected nodes.
//int   login_e_to_e:        Login end_to_end credit

/*****************************************************************/
//                          Temporary Variables                  //
/*****************************************************************/
//int          total_nodes:   Total number of nodes.
//Objid        temp_ids[200]:      Place to temp hold the Objids of the nodes
//char*        temp_char[10]:      Place to temp hold the name of the node
//char*        subSystem[10]: The subsystem of the node (EW, NAV, RDR etc.).
//int          what_subsys:   What subsytem am I a part of

//end computers
/*****************************************************************/
```

11. TO OTHERS STATE

The following code is contained in the **to others** state in the Fibre Channel node FSM.

```
//This state helps the processors decide which other nodes in the system they will
//talk to and builds the communications lists.
// count the nodes
total_nodes = op_topo_object_count (OPC_OBJMTYPE_NODE);
/*****************************************/
// initalize the arrays to some bogus Objid value like -99
for (i = 0; i < 200 ; i++)        {
```

```
        subnet_list[i]  = -99;
        d_id_list[i]    = -99;
        e_to_e[i]       = -99;
}
/*****************************************/
printf("the total number of nodes in the system is %d \n",total_nodes);
/*************************************************************/
//  This loads all the nodes into a temporary array
for (i = 0; i< total_nodes; i++)          {
        temp_ids[i] = op_topo_object(OPC_OBJMTYPE_NODE, i);
        printf("temp_ids[%d] is %d\n",i,temp_ids[i]);
}  // end for
/*************************************************************/
switch (what_subsys) {
        case 1: {               //If its in the EW subsys
                j=0;    //initialize the counter j
                for (i = 0; i< total_nodes; i++)          {

                //get the model name
                op_ima_obj_attr_get   (temp_ids[i], "model", &tmp_char);
                                if ( strcmp (tmp_char,"FC_proc" ) == 0 ||
                                                strcmp (tmp_char, "FC_ant" ) == 0 )
{       //if it's a computer or ant/sensor
                op_ima_obj_attr_get   (op_topo_child(temp_ids[i],
                  OPC_OBJTYPE_PROC, 0), "Subsystem", &subSystem);
                        if ( strcmp (subSystem, "EW") == 0 ||
                                strcmp(subSystem, "DISP") == 0 )    {
//and it's an EW computer, EW antenna, or display computer add it to the list
                op_ima_obj_attr_get(op_topo_child(temp_ids[i],
                  OPC_OBJTYPE_PROC, 0),"End to End", &login_e_to_e[j]);
                e_to_e[j] = login_e_to_e[j];
                subnet_list[j]   = op_topo_parent(temp_ids[i]);  //load the subnet
```

```
            d_id_list[j]     = temp_ids[i];                    //load the did
            j++;
                                    }
                                } //end if
                        } //end for
            break;
}//end case 1


case 2: {                   //if its part of the EO subsys
            j=0;    //initialize the counter j
            for (i = 0; i< total_nodes; i++)          {
//get the model name
            op_ima_obj_attr_get  (temp_ids[i], "model", &tmp_char);
                    if ( strcmp (tmp_char,"FC_proc" ) == 0 ||
                        strcmp (tmp_char, "FC_ant" ) == 0 )      {
            //if it's a computer or ant/sensor
                    op_ima_obj_attr_get (op_topo_child (temp_ids[i],
                      OPC_OBJTYPE_PROC, 0), "Subsystem", &subSystem);
                        if ( strcmp (subSystem, "EO") == 0 ||
                            strcmp (subSystem, "DISP") == 0 )          {
                        //and it's an EO computer, EO sensor
                        //or DISP proc add it to the list.
                        op_ima_obj_attr_get(op_topo_child (temp_ids[i],
                            OPC_OBJTYPE_PROC, 0), "End to End",
                                &login_e_to_e[j]);
                        e_to_e[j] = login_e_to_e[j];
                        subnet_list[j]  = op_topo_parent(temp_ids[i]);
                        d_id_list[j]    = temp_ids[i]; //load the did
                        j++;
                                }
                        } //end if
                    } //end for
```

```
        break;
}//end case 2


case 3: {               //If its part of the NAV subsys
        j=0;    //initialize the counter j
        for (i = 0; i< total_nodes; i++)        {
        //get the model name
        op_ima_obj_attr_get  (temp_ids[i], "model", &tmp_char);
                if ( strcmp (tmp_char,"FC_proc" ) == 0 ||
                  strcmp (tmp_char, "FC_ant" ) == 0 )       {
               //if it's a computer or ant/sensor
                  op_ima_obj_attr_get (op_topo_child (temp_ids[i],
                   OPC_OBJTYPE_PROC, 0), "Subsystem", &subSystem);
                    if ( strcmp (subSystem, "NAV == 0 ||
                                strcmp(subSystem, "DISP") == 0 )    {
                    //and it's an Nav computer, Nav sensor, or
                    //display computer add it to the list
                  op_ima_obj_attr_get(op_topo_child (temp_ids[i],
                     OPC_OBJTYPE_PROC, 0), "End to End",
                        &login_e_to_e[j]);
                        e_to_e[j] = login_e_to_e[j];
                        subnet_list[j]  = op_topo_parent(temp_ids[i]);
                        d_id_list[j]    = temp_ids[i];   //load the did
                                j++;
                        }
                        } //end if
                } //end for
        break;
}//end case 3
```

```
case 4: {                   //RDR
        j=0;    //initialize the counter j
        for (i = 0; i< total_nodes; i++)          {
        //get the model name
        op_ima_obj_attr_get   (temp_ids[i], "model", &tmp_char);
                if ( strcmp (tmp_char,"FC_proc" ) == 0 ||
                        strcmp (tmp_char, "FC_ant" ) == 0 ) {
                //if it's a computer or ant/sensor
                  op_ima_obj_attr_get (op_topo_child (temp_ids[i],
                   OPC_OBJTYPE_PROC, 0), "Subsystem", &subSystem);
                        if ( strcmp (subSystem, "RDR") == 0 ||
                                strcmp(subSystem, "DISP") == 0 )    {
                        //and it's an RDR computer RDR ant, or
                       //display computer add it to the list
        op_ima_obj_attr_get(op_topo_child (temp_ids[i],
        OPC_OBJTYPE_PROC, 0), "End to End", &login_e_to_e[j]);
        e_to_e[j] = login_e_to_e[j];
        subnet_list[j]  = op_topo_parent(temp_ids[i]);
        d_id_list[j]    = temp_ids[i];      //load the did
                                        j++;
                                }
                        }  //end if
                }   //end for
        break;
}//end case 4


case 5: {                   //DISP
        j=0;    //initialize the counter j
        for (i = 0; i< total_nodes; i++)          {
        //get the model name
        op_ima_obj_attr_get   (temp_ids[i], "model", &tmp_char);
```

```
                    if ( strcmp (tmp_char,"FC_proc" ) == 0 ||
                      strcmp (tmp_char, "FC_disp" ) == 0 )        {
                            //if it's a computer or display
                            op_ima_obj_attr_get   (op_topo_child (temp_ids[i],
                              OPC_OBJTYPE_PROC, 0), "Subsystem", &subSystem);
                                  if ( strcmp (subSystem, "DISP") == 0 )        {
                            //if it's a DISP computer or display
                                  op_ima_obj_attr_get(op_topo_child (temp_ids[i],
                                    OPC_OBJTYPE_PROC, 0), "End to End",
                                    &login_e_to_e[j]);
                                  e_to_e[j] = login_e_to_e[j];
                                  subnet_list[j]  = op_topo_parent(temp_ids[i]);
                                  d_id_list[j]    = temp_ids[i];          //load the did
                                              j++;
                                        }
                                  } //end if
                            }  //end for
                      break;
               }//end case 5
}//end switch
/*************************************************************************/
      j=0;    //initialize j
/*************************************************************************/
      //This just prints out the list of processers in (Sub_net id, d_id) form
      while (op_topo_parent(d_id_list[j]) != OPC_OBJID_INVALID)    {
            printf("(%d , %d) \n",subnet_list[j], d_id_list[j]);
            j++;
      }              //end while
      num_proc = j;
      printf("Others J is equal to %d \n",j);
```

```
/*****************************************************************/
//                          Global Variables                     //
/*****************************************************************/
//                              NONE
/*****************************************************************/
//                          State Variables                      //
/*****************************************************************/
        //int   d_id_list[200]:     Destinations I might talk to
        //int   subnet_list[200]:   Their assoc. subnets
        //int   e_to_e[200]:        End_to_end credit of the connected nodes.
        //int   login_e_to_e:       Login end_to_end credit


/*****************************************************************/
//                          Temporary Variables                  //
/*****************************************************************/
//int           total_nodes:    Total number of nodes.
//Objid         temp_ids[200]:      Place to temp hold the Objids of the nodes
//char*         temp_char[10]:      Place to temp hold the name of the node
//char*         subSystem[10]:  The subsystem of the node (EW, NAV, RDR etc.).
//int           what_subsys:    What subsytem am I a part of

//end others
/*****************************************************************/
```

12. TOPO_BUILD STATE

The following code is contained in the **topo_build** state in the Fibre Channel node FSM.

```
/*****************************************************************/
//Topo_build updates the entire topology on a
//scheduled basis set by global var DEL_T
// reset the route pointers to null

        for (i = 0; i < 200; i++) {
                rte_arry[i] = NULL;
        }
        //Am I the first process to wake up?
        if (LAST_TOPO_UPDATE_TIME < op_sim_time())      {
                LAST_TOPO_UPDATE_TIME = op_sim_time(); //update Last update
                TOPO_PTR = op_rte_topo_from_objids();  //update topo_ptr
                op_intrpt_schedule_self(op_sim_time() + DEL_T, 0);
        }
/*****************************************************************/
//                              Global Variables                 //
/*****************************************************************/
//Topology*   TOPO_PTR:  The only Topology Pointer
//double      LAST_TOPO_UPDATE_TIME:  When was the topology last updated

/*****************************************************************/
//                              State Variables                  //
/*****************************************************************/
//Route*      rte_arry[200]:  Holds routes from this node to all others.

/*****************************************************************/
//                              Temporary Variables              //
/*****************************************************************/
//                                  NONE

// end topo_build
/*****************************************************************/
```

13. GET_PKT STATE

The following code is contained in the **get_pkt** state in the Fibre Channel node FSM.

```
/*****************************************************************/
//get_pkt is responsible for getting packets that
//originated in the ideal packet generator source
pkptr = op_pk_get (SRC_IN_STRM);
/*****************************************************************/
//Get an integer which is uniformly distributed
//do while did = sid (don't want to send to yourself)

do      {
        //uniform number cast as an int from zero to num proc which is 1 more than index
        dest_index = (int)op_dist_uniform (num_proc);
        printf("The index is %d \n",dest_index);
        dst_subnet = subnet_list[dest_index];
        d_id = d_id_list[dest_index];
                printf("My destination is (%d , %d) \n", dst_subnet, d_id);
        }       while (d_id == s_id);   //make sure you're not sending it to yourself.

//end do/while

/*****************************************************************/
//                          Global Variables                     //
/*****************************************************************/
//                              NONE
```

```
/*****************************************************************/
//                          State Variables                      //
/*****************************************************************/
//int    d_id_list[200]:  Destinations I might talk to
//int    subnet_list[200]:  Their assoc. subnets
//int    num_proc:                Number of nodes in the system
/*****************************************************************/
//                          Temporary Variables                  //
/*****************************************************************/
//Packet*     pkptr:   The pointer to the active packet.
//int         dest_index:  The index used to access the arrays.
//int         dst_subnet:     The subnet assoc with the destination
//int         d_id:           The destination.

// end get_pkt
/*****************************************************************/
```

14. B_TO_B STATE

The following code is contained in the **b_to_b** state in the Fibre Channel node FSM.

```
/*****************************************************************/
//This state checks the buffer to buffer credit
//if you have no buffer to buffer credit the statisic
//is logged and the buffer refilled
//if credit is avail, it is decremented and the pkt
//fwd to b_plate

if (buf_to_buf == 0)   {  //     is buf_to_buf = 0?
        no_bb_count = no_bb_count++;            //increment the number of times
                                                //that you've reset the bb credit.
```

```
        op_stat_write(no_bb_count_sh, no_bb_count);  //store the statistic
        buf_to_buf = max_buf;                        //reset the buffer
}
/*****************************************************************/
//                          Global Variables                     //
/*****************************************************************/
//                             NONE
/*****************************************************************/
//                          State Variables                      //
/*****************************************************************/
//Stathandle    no_bb_count_sh: SH to the bb_count stat.
//int           buf_to_buf:     Buf_to_buf credits.
//int           max_buf:        Login value of buf_to_buf credits
//int           no_bb_count:    Count of the number of times b_to_b is reset

/*****************************************************************/
//                          Temporary Variables                  //
/*****************************************************************///
                                NONE
//end b_to_b
/*****************************************************************///
```

15. B_PLATE STATE

The following code is contained in the **b_plate** state in the Fibre Channel node FSM.

```
/*****************************************************************///
//b_plate sets standard fields in the header and sets the
//size of the data payload and finally sets the class
//of the message
/*****************************************************************/
/*      The following is Frame boiler_plate                      */
/*****************************************************************/
```

```
// These settings are the same no matter the class of the
//message

op_pk_nfd_set(pkptr, "SOF_0", 0xFA);  //K28.5
op_pk_nfd_set(pkptr, "SOF_1", 0xB5);  //D21.5
op_pk_nfd_set(pkptr, "R_Bits", 2);
op_pk_nfd_set(pkptr, "INFO", 2);
op_pk_nfd_set(pkptr,"D_ID", d_id);
op_pk_nfd_set(pkptr, "S_ID", s_id);

//        This sets the src id field which is useful for sending
//rrdy's back to the guy who sent you the frame.  src id
//has zero (0) length and does not affect throughput calc.

op_pk_nfd_set (pkptr, "src id", s_id);
op_pk_nfd_set(pkptr, "src subnet", subnet);

//There is no rx_id yet so set to ffff

op_pk_nfd_set(pkptr, "RX_ID", 0xffff);

//Use attr_get to get the extended attribute Data Size
// Data Size is the mean data size sent by the node.

node_id = op_id_self();
op_ima_obj_attr_get(node_id, "Data Size", &data_size);

//Protocol is the FC-AE protocol so set Type to 0x4a per p. 188//

op_pk_nfd_set(pkptr, "Type", 0x4a);

//set the data field size, if you want MAX_DATA_SIZE for all Frames
//Set the attribute Data Size to -99.
```

```
        if(data_size == -99)    {
                op_pk_bulk_size_set(pkptr, MAX_DATA_SIZE);
        }       else    {
                op_pk_bulk_size_set(pkptr, data_sz_set(data_size));
        } //end if

//set the creation time
op_pk_nfd_set(pkptr, "Parameter", op_sim_time());

//What class is the msg??

class_type = op_dist_outcome(class_dist);
printf("The Class I get is %d\n",class_type);

/*****************************************************************/
//                          Global Variables                     //
/*****************************************************************/
//const int     MAX_DATA_SIZE:  In bits, the largest the data payload can be.
/*****************************************************************/
//                          State Variables                      //
/*****************************************************************/

//int           s_id:           The source id of this node
//int           subnet:         The subnet of this node
/*****************************************************************/
//                          Temporary Variables                  //
/*****************************************************************/

//Packet*       pkptr:          The pointer to the active packet.
//Objid         node_id:        The Objid of this node.
//int           d_id:           The destination.
//int           data_size:  The mean size(in words) of the node.
//int           class_type: The class of the message
/*****************************************************************/
// end b_plate
/*****************************************************************/
```

16. CL1_XMT STATE

The following code is contained in the **CL1_xmt** state in the Fibre Channel node FSM.

```
/*****************************************************************/
//This state would be used to format class 1 messages
//class 1 is not implemented yet (Aug 00)
/*****************************************************************/
/*      The following is SOFi1 p111 fc man                       */
/*****************************************************************/
op_pk_nfd_set(pkptr, "SOF_2", 0x57);  //D23.2
op_pk_nfd_set(pkptr, "SOF_3", 0x57);  //D23.2

printf("CLASS 1\n");

//CS_CTL is set  in class 1//
op_pk_nfd_set(pkptr,"CS_CTL", 0x00);

/*****************************************************************/
//                          Global Variables                     //
/*****************************************************************/
//                            NONE
/*****************************************************************/
//                          State Variables                      //
/*****************************************************************/
//                            NONE
/*****************************************************************/
//                          Temporary Variables                  //
/*****************************************************************/
//Packet*       pkptr:   The pointer to the active packet.
//end CL1_xmt
/*****************************************************************/
```

17. CL2_XMT STATE

The following code is contained in the **CL2_xmt** state in the Fibre Channel node FSM.

```
/*****************************************************************/
//This state is used for formatting Class 2 messages
/*****************************************************************/
/*      The following is SOFi2 p111 fc man                       */
/*****************************************************************/
op_pk_nfd_set(pkptr, "SOF_2", 0x55);
op_pk_nfd_set(pkptr, "SOF_3", 0x55);

printf("CLASS 2\n");

//CS_CTL is not used in class 2//
op_pk_nfd_set(pkptr,"CS_CTL", 0xff);

/*****************************************************************/
//                        Global Variables                       //
/*****************************************************************/
//                          NONE
/*****************************************************************/
//                        State Variables                        //
/*****************************************************************/
//                          NONE
/*****************************************************************/
//                        Temporary Variables                    //
/*****************************************************************/
//Packet*       pkptr:   The pointer to the active packet.

//end CL2_xmt
/*****************************************************************/
```

18. CL3_XMT STATE

The following code is contained in the **CL3_xmt** state in the Fibre Channel node FSM.

```
/*****************************************************************/
//This state is used for formatting Class 2 messages
/*****************************************************************/
/*      The following is SOFi3 p111 fc man          */
/*****************************************************************/
op_pk_nfd_set(pkptr, "SOF_2", 0x56);  //D22.2
op_pk_nfd_set(pkptr, "SOF_3", 0x56);  //D22.2

printf("CLASS 3\n");

//CS_CTL is not used in class 3//
op_pk_nfd_set(pkptr,"CS_CTL", 0xff);

/*****************************************************************/
//                          Global Variables                     //
/*****************************************************************/
//                            NONE
/*****************************************************************/
//                          State Variables                      //
/*****************************************************************/
//                            NONE
/*****************************************************************/
//                          Temporary Variables                  //
/*****************************************************************/
//Packet*       pkptr:   The pointer to the active packet.

//end CL3_xmt
/*****************************************************************/
```

19. E_TO_E STATE

The following code is contained in the **e_to_e** state in the Fibre Channel node FSM.

```
/*****************************************************************/
//e_to_e checks the end to end credit with the far side node
//if there is no credit the statistic is logged and credit
//is restored to its original login value.

if (e_to_e[dest_index] == 0)  {                  //is end_to_end = 0?
        no_ee_count = no_ee_count++;
        op_stat_write(no_ee_count_sh, no_ee_count );
        //store the statistic
        e_to_e[dest_index] = login_e_to_e[dest_index];
        //reset the credit
}
e_to_e[dest_index] = e_to_e[dest_index]--;  //decrement the end_to_end credit

/*****************************************************************/
//                          Global Variables                     //
/*****************************************************************/
//                            NONE
/*****************************************************************/
//                          State Variables                      //
/*****************************************************************/
//Stathandle    no_ee_count_sh: SH to the ee_count stat.
//int           no_ee_count:  Count of the number of times e_to_e is reset
//int           e_to_e[200]:   End_to_end credit of the connected nodes.
//int           login_e_to_e:          Login end_to_end credit
```

```
/*****************************************************************/
//                        Temporary Variables                    //
/*****************************************************************/

//int           dest_index:  The index used to access the arrays.

//end e_to_e
/*****************************************************************/
```

20. ROUTE STATE

The following code is contained in the **route** state in the Fibre Channel node FSM.

```
/*****************************************************************/
//route is responsible for adding the route to the packet
//First check to see if the rte_arry[] is null
//This will tell you if the route has been built already, otherwise use the SV
//rte_arry[dest_index] to make the tmp_rte which is added to the packet

if (rte_arry[dest_index] == NULL)    {
        rset_ptr = op_rte_routeset_create (TOPO_PTR, subnet, s_id,
                dst_subnet, d_id, MAX_HOPS);
        tmp_rte = op_rte_routeset_mincost (rset_ptr);
        op_rte_route_print (tmp_rte);
        rte_arry[dest_index] =  op_rte_route_copy (tmp_rte);
        op_rte_route_print (rte_arry[dest_index]);
        //destroy the now useless route set pointer
        op_rte_routeset_destroy(rset_ptr);
}
```

```
op_rte_route_print (rte_arry[dest_index]);
tmp_rte =  op_rte_route_copy (rte_arry[dest_index]);
op_rte_pk_route_insert(pkptr, tmp_rte);          //add the route to the packet

/*****************************************************************/
//                          Global Variables                     //
/*****************************************************************/
//Topology*    TOPO_PTR:  The only Topology Pointer
//const int    MAX_HOPS:  Max number of Hops a path can have.
/*****************************************************************/
//                          State Variables                      //
/*****************************************************************/
//Route*       rte_arry[200]:  Holds routes from this node to all others.
//int          s_id:           The source id of this node
//int          subnet:         The subnet of this node
/*****************************************************************/
//                          Temporary Variables                  //
/*****************************************************************/
//Route_Set*   rset_ptr:  Pointer to the routeset.
//Route*             tmp_rte:   A copy of the route to be used.
//int                dst_subnet:    The subnet assoc with the destination
//int                d_id:          The destination.
//int                dest_index:  The index used to access the arrays.

//end route
/*****************************************************************/
```

21. XMT_PKT STATE

The following code is contained in the **xmt_pkt** state in the Fibre Channel node FSM.

```
/*****************************************************************/
//xmt_pkt is responsible for sending the packet
//out of the node with or without delay
```

```
buf_to_buf--;          //decrement the buffer to buffer

if (proc_delay != 0 )   {              //if the process delay is not zero then

//send it out the transmitter with the proscribed delay
//  the following code allows a PDF (exponential is
//  used but any PDF could be put here) to be applied
//  to the delay

//        op_pk_send_delayed (pkptr, XMT_OUT_STRM,
//                                op_dist_exponential (proc_delay));

//  For testing and verification a constant delay was used
        op_pk_send_delayed (pkptr, XMT_OUT_STRM, proc_delay);
} else {
//send it with no delay
        op_pk_send (pkptr, XMT_OUT_STRM);
}

/*************************************************************************/
//                              Global Variables                         //
/*************************************************************************/
//                                NONE
/*************************************************************************/
//                              State Variables                          //
/*************************************************************************/
//double   proc_delay:  The processing delay associated with this node
//int            buf_to_buf:     Buf_to_buf credits.
```

```
/**************************************************************************/
//                          Temporary Variables                           //
/**************************************************************************/
//Packet*       pkptr:   The pointer to the active packet.

//end xmt_pkt
/**************************************************************************/
```

22. RCV STATE

The following code is contained in the **rcv** state in the Fibre Channel node FSM.

```
/**************************************************************************/
//rcv state is used when a packet is received
//on the node receiver (as opposed to the pkt generator)
//state figures out if pkt is a frame or primative

numFields = -1;  //init to ensure proper xsitions
c3Rrdy = -1;

/*  get the packet                                          */
pkptr = op_pk_get (RCV_IN_STRM);

op_pk_format_info_get(pkptr,
                OPC_PK_PROPERTY_NUMBER_FIELDS, &numFields);

if (numFields < 3 && numFields > 0)         {  //if less than 3 it is an rrdy
        c3Rrdy = 1;
}       else    {
        c3Rrdy = 0;  //if > 3 it is a frame
}
```

```
/*****************************************************************/
//                          Global Variables                     //
/*****************************************************************/
//                              NONE
/*****************************************************************/
//                          State Variables                      //
/*****************************************************************/
//                              NONE
/*****************************************************************/
//                          Temporary Variables                  //
/*****************************************************************/
//Packet*       pkptr:   The pointer to the active packet.
//int           numFields:      The number of fields in the frame.
//int           c3Rrdy:                 Indicates the reception of an R_Rdy

//end rcv
/*****************************************************************/
```

23. R_RDY_RCV STATE

The following code is contained in the **r_rdy_rcv** state in the Fibre Channel node FSM.

```
/*****************************************************************/
//r_rdy_rec is used when a primative is received
//if an r_rdy was received the b_to_b credit is icremented
//provided that the credit isn't at max already

if (buf_to_buf < max_buf) {   //if the buffer isn't maxed out
        buf_to_buf++;                       //add one
}       else    {
        buf_to_buf = max_buf;       //  otherwise leave it a max, because credit can't
                                    //be greater than the max.
}
```

```
op_pk_destroy(pkptr);

/*****************************************************************/
//                          Global Variables                     //
/*****************************************************************/
//                             NONE
/*****************************************************************/
//                          State Variables                      //
/*****************************************************************/
//int           buf_to_buf:     Buf_to_buf credits.
//int           max_buf:        Login value of buf_to_buf credits
/*****************************************************************/
//                          Temporary Variables                  //
/*****************************************************************/
//Packet*       pkptr:  The pointer to the active packet.

//end r_rdy_rec
/*****************************************************************/
```

24. SEND_RRDY STATE

The following code is contained in the **send_rrdy** state in the Fibre Channel node FSM.

```
/*****************************************************************/
//snd rrdy builds an rrdy and sends it back to the previous node
//that sent the frame.

printf("I got a frame!\n");

rdyptr = op_pk_create_fmt ("R_RDY");
```

```
//get the src node from the frame and put it into src_nde
op_pk_nfd_get (pkptr, "src id", &src_nde);

//put your s_id in the "src id" field
op_pk_nfd_set (rdyptr, "src_nde", s_id);

//calculate the delay
//check for delay
if (r_rdy_delay != 0 )  {

//send it out the transmitter with the proscribed delay
//        op_pk_send_delayed (rdyptr, XMT_OUT_STRM,
//                                op_dist_exponential (r_rdy_delay));      //delay with some
distribution
        op_pk_send_delayed (rdyptr, XMT_OUT_STRM, r_rdy_delay);   //constant delay
} else {
//send it with no delay
        op_pk_send (rdyptr, XMT_OUT_STRM);
}

//        get it's R_Bits and assign it to fc_type
//so that we can transition to the next state

op_pk_nfd_get(pkptr,"R_Bits", &fc_type);
printf("Done with snd_rrdy\n");

/*************************************************************************/
//                                Global Variables                                //
/*************************************************************************/
//                                NONE
```

```
/*****************************************************************/
//                          State Variables                      //
/*****************************************************************/

//double   proc_delay: The processing delay associated with this node
//int          s_id:        The source id of this node
/*****************************************************************/
//                          Temporary Variables                  //
/*****************************************************************/
//Packet*      pkptr:       The pointer to the active packet.
//Packet*      rdyptr:      The pointer to the created R_RDY packet.
//int          src_nde:     Put into the src_nde field for ease of return
//int          fc_type:     Is it FT-1 or FT-0?


//end snd_rrdy
/*****************************************************************/
```

25. FC_0 STATE

The following code is contained in the **FC_0** state in the Fibre Channel node FSM.

```
/*****************************************************************/
//The FC_0 state is entered when a frame type zero
//message arrives at the node (FT-0 are link control frames)
//Actions are taken based on the type of LCF that is
//received.

// find out where it came from and increment that
//value of e_to_e[i]

op_rte_pk_src_node(pkptr, &src_subnet, &src_id);
        for (k = 0; k < 200; k++)       {
                if (d_id_list[k] == src_id)     {
                        printf("E to E for %d is %d\n", src_id, e_to_e[k]);
```

```
                    e_to_e[k] = e_to_e[k]++;
                    break;
                }

        }

printf("E to E for %d is now %d\n", src_id, e_to_e[k]);

op_pk_nfd_get(pkptr,"Parameter", &create_time);
ete_delay = op_sim_time () - create_time;   // calculate the end to end delay
op_stat_write (ete_gsh, ete_delay);          // write the statistic

//find out if it is an F_BSY or not
op_pk_nfd_get(pkptr, "INFO", &f_bsy);

if (f_bsy == 0x5)       { //if it's an f_bsy to a data frame
        F_BSY_DAT = F_BSY_DAT++;  //increment the stat
        //printf("F_BSY_DAT IS %d \n", f_bsy_dat);
        op_stat_write (f_bsy_dat_gsh, F_BSY_DAT);  //write the stat

}       else if (f_bsy == 0x6) { //it's an f_bsy to an LCF
                F_BSY_LCF = F_BSY_LCF++; // increment the stat
                //printf("F_BSY_LCF IS %d\n", f_bsy_lcf);
                op_stat_write (f_bsy_lcf_gsh, F_BSY_LCF);  //write the stat

        } else if (f_bsy == 0x2)   {  // the frame got there and back so calc e-to-e delay
                        op_pk_nfd_get(pkptr,"Parameter", &create_time);
                        ete_delay = op_sim_time () - create_time;
                        printf("Create Time = %f , Sim Time = %f, Delay = %f \n",
                                create_time, op_sim_time(), ete_delay);
                        op_stat_write (ete_gsh, ete_delay);
                        }
```

```
op_pk_destroy(pkptr);

/*****************************************************************/
//                          Global Variables                     //
/*****************************************************************/
//int       F_BSY_DAT:      Holds data for F_BSY's to data frames.
//int       F_BSY_LCF:      Holds data for F_BSY's to LCF frames.
/*****************************************************************/
//                          State Variables                      //
/*****************************************************************/

//Stathandle    ete_gsh:        SH to the ete global stat.
//Stathandle    f_bsy_dat_gsh:  SH to the f_bsy to data stat.

//Stathandle    f_bsy_lcf_gsh:  SH to the f_bsy to LCF stat.
//int   d_id_list[200]:     Destinations I might talk to

//int   e_to_e[200]:
End_to_end credit of the connected nodes.
/*****************************************************************/
//                          Temporary Variables                  //
/*****************************************************************/
//Packet*       pkptr:      The pointer to the active packet.
//int           src_id:         The source node.
//int           src_subnet:     The src_id's assoc subnet.
//int           f_bsy:          Used to check if the packet was an F_BSY
//double        create_time:    The time the packet was created.
//double        ete_delay:      The time it took to get here.

//end FC_0
/*****************************************************************/
```

26. FC_1 STATE

The following code is contained in the **FC_1** state in the Fibre Channel node FSM.

```
/*****************************************************************/
//FC_1 state is entered when a data frame (FT-0) is
//received by the node
//This state returns acks if nec, otherwise it calculates
//end to end delay for class 3 msgs.
// init the transition variable
        class_3_rec = -1;

op_pk_nfd_get(pkptr,"SOF_3",&classType);
        printf("%d\n",classType);

        if (classType == 0x56)          {   //if it's class 3 no ack is rqd
                //ete_delay = op_sim_time () - op_pk_creation_time_get(pkptr);
                //op_stat_write (ete_gsh, ete_delay);
                printf("here\n");
                op_pk_nfd_get(pkptr,"Parameter", &create_time);
                ete_delay3 = op_sim_time () - create_time;
                op_stat_write (ete_gsh_cl3, ete_delay3);
                op_pk_destroy(pkptr);
                class_3_rec = 1;
        } else if (classType == 0x57 || classType == 0x55)  {
                //else if it's class 1 or 2 return an ack
                //where it came from and where the fc-0 has to go back to
                printf("I'm returning the fc-0 now\n");

                op_rte_pk_src_node(pkptr, &dst_subnet, &d_id);
                        for (k = 0; k < 200; k++)       {
                        if (d_id_list[k] == d_id)       {
                                dest_index = k;
                                break;
                        }
                }
```

```
                //build the ACK for sending
                op_pk_nfd_set(pkptr, "D_ID", d_id);
                op_pk_nfd_set(pkptr, "S_ID", s_id);
                op_pk_nfd_set(pkptr, "R_Bits", 0xC); //make it an ack
                op_pk_nfd_set(pkptr, "RX_ID", 0x00); //give it an exchange ID
                op_pk_bulk_size_set(pkptr,0);   //make the data size zero
                op_pk_nfd_set(pkptr, "src id", s_id);  //for sending rrdys


        class_3_rec = 0;
        } //end else
/*************************************************************************/
//                              Global Variables                         //
/*************************************************************************/
//                                  NONE
/*************************************************************************/
//                              State Variables                          //
/*************************************************************************/
//Stathandle    ete_gsh_cl3:   SH to the ete_cl3 global stat.
//int     d_id_list[200]:          Destinations I might talk to
//int     s_id:                        The source id of this node
/*************************************************************************/
//                              Temporary Variables                      //
/*************************************************************************/
//Packet*       pkptr:         The pointer to the active packet.
//double        create_time:   The time the packet was created.
//double        ete_delay3:          The time it took to get here.
//int           dest_index:    The index used to access the arrays.
//int           dst_subnet:          The subnet assoc with the destination
//int           d_id:                The destination.
//int           class_3_rec:   Was a class 3 received or not.
//int           classType:           Holds the class of the frame.
//end fc-1
/*************************************************************************/
```

27. CHECK CREDIT STATE

The following code is contained in the **check credit** state in the Fibre Channel node FSM.

```
/*****************************************************************/
//check credit makes sure that there is adequate b_to_b
//credit avail to send the ACK back
//if not, the b2b stat is incremented and b2b credit restored

if (buf_to_buf == 0)   {  //     is buf_to_buf = 0?
        no_bb_count = no_bb_count++;              //increment the number of times
                                                  //that you've reset the bb credit.
        op_stat_write(no_bb_count_sh, no_bb_count);  //store the statistic
        buf_to_buf = max_buf;                 //reset the buffer
}
/*****************************************************************/
//                          Global Variables                     //
/*****************************************************************/
//                             NONE
/*****************************************************************/
//                          State Variables                      //
/*****************************************************************/

//Stathandle    no_bb_count_sh: SH to the bb_count stat.
//int           no_bb_count:   Count of the number of times b_to_b is reset
//int           buf_to_buf:      Buf_to_buf credits.
//int           max_buf:         Login value of buf_to_buf credits
/*****************************************************************/
//                          Temporary Variables                  //
/*****************************************************************/
//                             NONE

//end check credit
/*****************************************************************/
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. OPNET SWITCH FINITE STATE MACHINE CODE

1. LIST OF STATE VARIABLES

Table C-1. List of State Variables Used in the Switch Finite State Machine

Variable Name	Variable Type	States where variable is used
s_id	int	objid?, snd rrdy, xmt
subnet	int	subnet?, add_route
buf_to_buf[32]	int	r_rdy, init
xmt_id[32]	int	con_out
link_id[32]	Objid	r_rdy, sndrrdy, xmt, con_out, dec_buf
node_id[32]	int	r_rdy, sndrrdy, xmt, con_out, dec_buf
util[32]	float	init, set_link_cost
total_bits	double	sndrrdy, xmt, init
rte_arry	Route *	add_route
rset_ptr	Route_Set *	add_route
r_rdy_delay	double	objid?, sndrrdy
proc_delay	double	objid?, xmt
max_credit	int	r_rdy, init
ded_cl_3_gsh	Stathandle	init, dec_buf

2. LIST OF TEMPORARY VARIABLES

Table C-2. List of Temporary Variables Used in the Switch Finite State Machine

Variable Name	Variable Type	States where variable is used
temp_node_id	Objid *	objid?
node_type[10]	char *	objid?
node_name[10]	char *	objid?
subnet_id	Objid *	subnet?
xmt_name[5]	char	con_out
pkptr	Packet *	type, r_rdy, frame, sndrrdy, xmt, f_bsy, add_route
numFields	int	type

streamIn	int	type
src_nde	int *	r_rdy, sndrrdy
inc_buf	int	r_rdy
subnet_id_ptr	int *	frame
node_id_ptr	int *	frame
next_subnet	int *	frame, add_route
next_node	int *	frame, xmt, dec_buf, add_route
dec_buf	int	dec_buf
credit	int	dec_buf
class3	int	dec_buf
classType	int	dec_buf
rdyptr	Packet *	sndrrdy
streamOut	int	sndrrdy, xmt
fc_type	int	f_bsy
temp_sid	int	f_bsy
temp_did	int	f_bsy
add_route	int	f_bsy
tmp_rte	Route *	add_route
dst_subnet	int	add_route
d_id	int	add_route
i	int	index
j	int	index

3. HEADER BLOCK

The following code is contained in the header block of the node's process model.

```
/*****************************************************************/
//Define all transitions as macros

#define PK_ARRVL1 ( op_intrpt_type () == OPC_INTRPT_STRM )
#define RRDY (numFields < 3)
```

```
#define FRAME (numFields > 3)
#define UPDATE_COST (op_intrpt_type() == OPC_INTRPT_SELF)
#define CREDIT_AVAIL (credit == 1)
#define NO_CREDIT (credit == 0)
#define NO_CRED_CL_3 (class3 == 1)
/*****************************************************************/
//                          Global Variables                     //
/*****************************************************************/
        double      START_TIME = -1; //start time of the simulation
const   int                 BAND_WIDTH = 1062500000; //BW of Fibre channel
extern  Topology    *TOPO_PTR;      //The one and only Topology pointer.
extern  const int   MAX_HOPS;       //Max number of hops a packet can take.
        int         DED_CL_3;       //Holds the number of Class-3 packets
                                    //destroyed by switches.
const   double      DEL_COST = .1;  //How often you update the cost of the links.
/*****************************************************************/
//                          Function Prototypes                  //
/*****************************************************************/
//Function: get_outStream
//Purpose:  Returns the outstream  to reach the node of interest
int get_outStream(const int [], const Objid [], int nde);

//end Header Block
/*****************************************************************/
```

4. FUNCTION BLOCK

The following code is contained in the function block.

```
/*****************************************************************/
//Function: get_outStream
//Purpose:  Returns the outstream  to reach the node of interest
/*****************************************************************/
```

```
int get_outStream(const int nde_ary[], const Objid lnk_id [], int src_nde)   {
        int i = 0;  //counter
        int out;
        double  cost;
        double  low_cost = 100;

        for(i = 0; i <= 31; i++)          {
                if (nde_ary[i] == src_nde)      {
                        op_ima_obj_attr_get(lnk_id[i], "cost", &cost);
                        if (cost < low_cost){
                                low_cost = cost;
                                out = i;
                        }
                }
        }  //end for
return out;
} //end get_outStream
/*************************************************************************/
```

5. INIT STATE

The following code is contained in the **init** state in the Fibre Channel switch FSM.

```
/*************************************************************************/
//Init state is used to do initial admin tasks in the
//switch.  This would be like a login in Fibre Channel

//establish the buffer to buffer credit with each port.
op_ima_obj_attr_get(op_id_self (), "Max Credit", &max_credit);
printf("Max Credit = %d \n", max_credit);
for (i = 0; i <= 31; i++)          {
        buf_to_buf[i] = max_credit;
}
```

```
//set first link cost setting interrupt
op_intrpt_schedule_self(op_sim_time(),0);

if (START_TIME < op_sim_time()) {
        START_TIME = op_sim_time();

        //init the Global Statistic var's only one time!
        DED_CL_3 = 0;
}

//Register the statistic
ded_cl_3_gsh = op_stat_reg ("Dead Class 3",
                OPC_STAT_INDEX_NONE, OPC_STAT_GLOBAL );

for (i = 0; i < 32; i++) {          //init utiliztion and total bits
        util[i] = 0.0;
        total_bits[i] = 0.0;
}
/*****************************************************************/
//                          Global Variables                     //
/*****************************************************************/
//double        START_TIME:       The time the simulation started.
//int           DED_CL_3:   Stat for number of class 3 frames destroyed by the switch.
/*****************************************************************/
//                          State Variables                      //
/*****************************************************************/
//Stathandle  ded_cl_3_gsh:   SH which holds the value of DED_CL_3
//double     total_bits[32]: Total number of bits to pass through the assoc link.
//float      util[32]:      Utilization of the link.
//int              max_credit:     The login buf_to_buf credit.
//int              buf_to_buf[32]: Holds buf_to_buf credit of the attached nodes.
```

```
/*****************************************************************/
//                        Temporary Variables                    //
/*****************************************************************/
//                              NONE


//end init
//*****************************************************************//
```

6. OBJID? STATE

The following code is contained in the **objid?** state in the Fibre Channel switch FSM.

```
//*****************************************************************//
//objid? state determines the objid of this switch
//it also gets the delay parameters

temp_node_id = op_topo_parent(op_id_self());
s_id = temp_node_id;
printf("\nI am S_ID # %3d \n", s_id);
op_ima_obj_attr_get (temp_node_id, "model" , &node_type);
op_ima_obj_attr_get (temp_node_id, "name"  , &node_name);
printf("my name is %s \n", node_name);

//get the process and R_RDY delays
op_ima_obj_attr_get (op_id_self (), "Processing Delay"  , &proc_delay);
op_ima_obj_attr_get (op_id_self (), "R_RDY Delay"  , &r_rdy_delay);

printf("Proc delay = %f ", proc_delay);
printf("R_rdy delay = %f\n", r_rdy_delay);
```

```
/*****************************************************************/
//                          Global Variables                     //
/*****************************************************************/
//                              NONE
/*****************************************************************/
//                          State Variables                      //
/*****************************************************************/
//int       s_id:          The source id of this node
//double    proc_delay: The processing delay associated with this node
//double    r_rdy_delay: The delay assoc with sending an r_rdy
/*****************************************************************/
//                          Temporary Variables                  //
/*****************************************************************/
//Objid*    temp_node_id:The Objid of this node.
//char*     node_type[10]:  The type of node (i.e. proc, disp, ant)
//char*     node_name[10]:  The name of the node.

//end objid?
//*****************************************************************//
```

7. SUBNET? STATE

The following code is contained in the **subnet?** state in the Fibre Channel switch FSM.

```
//*****************************************************************//
//Subnet state determines the subnet of the switch
//This is used for the routing of packets and does not
//mirror any functionality of Fibre Channel

subnet_id = op_topo_parent(temp_node_id);
subnet = subnet_id;
printf("My subnet # is: %d \n", subnet);
```

```
/*****************************************************************/
//                          Global Variables                     //
/*****************************************************************/
//                             NONE
/*****************************************************************/
//                          State Variables                      //
/*****************************************************************/
//int           subnet:  The subnet of the assoc node.
/*****************************************************************/
//                          Temporary Variables                  //
/*****************************************************************/
//Objid*        subnet_id:  Temp holder for the subnet Objid.

//end subnet?
//*****************************************************************//
```

8. CON_OUT STATE

The following code is contained in the **con_out** state in the Fibre Channel switch FSM.

```
//*****************************************************************//
// This module finds all the links attached to the switch
//and builds a look-up table so when a packet is routed
//to a node, the switch can quickly find the outstream

for ( j=0; j<= 31; j++)          {
        sprintf(xmt_name, "xmt%d", j);  // all the xmitters have a std name
        printf("xmt_name = %s \n",xmt_name);
                //now we find the objid of the xmitter and save it
        xmt_id[j]=op_id_from_name(s_id, OPC_OBJTYPE_PTTX, xmt_name);
        printf("xmt_id is %d \n",xmt_id[j]);
```

```
//find the objid of the attached link and save it
        link_id[j] = op_topo_assoc( xmt_id[j], OPC_TOPO_ASSOC_OUT,
                                                OPC_OBJTYPE_LKSIMP, 0);
        printf("link_id is %d \n",link_id[j]);

//  if the link id is valid (ie if the xmitter is connected to a node)
//then the node_id is the parent of the receiver connected to the link
                if (link_id[j] != OPC_OBJID_INVALID)    {
                    node_id[j] = op_topo_parent( op_topo_assoc(link_id[j],
                        OPC_TOPO_ASSOC_OUT, OPC_OBJMTYPE_RECV, 0));
                }       //end if

printf("The sid, link and connected nodes are  %d , %d , %d \n",
        s_id, link_id[j], node_id[j]);
}
/*****************************************************************/
//                          Global Variables                     //
/*****************************************************************/
//                              NONE
/*****************************************************************/
//                          State Variables                      //
/*****************************************************************/
//int       xmt_id[32]:  Objid's of all the xmitters on the switch
//int       node_id[32]: Objid's of all the nodes attached to the switch.
//int       link_id[32]:  Objid's of all the links attached to the switch.
/*****************************************************************/
//                          Temporary Variables                  //
/*****************************************************************/
//char   xmt_name[5]:  Name of the xmitter on the switch.

//end con_out
//*****************************************************************//
```

9. SET_LINK_COST STATE

The following code is contained in the **set_link_cost** state in the Fibre Channel switch FSM.

```
//*********************************************************************//
//set_link_cost is responsible for keeping the "cost"
//associated with traversing a link.  This will ensure
//that over time the traffic is balanced.

op_intrpt_schedule_self(op_sim_time() + DEL_COST, 0);  //schedule the next intrpt

for (i = 0; i < 32; i++) {
        //printf("link_id %d\n",link_id[i]);
        if (link_id[i] != OPC_OBJID_INVALID)    {
                util[i] = (total_bits[i]/(op_sim_time() - START_TIME))/BAND_WIDTH;
                op_ima_obj_attr_set (link_id[i], "cost", util[i]);
                //printf("util is %f \n",util[i]);
        }
}
/*********************************************************************/
//                              Global Variables                     //
/*********************************************************************/
//double        START_TIME:     The time the simulation started.
//const double  DEL_COST:       How often the cost is updated.
//const int     BAND_WIDTH:     The bandwidth of Fibre Channel.
/*********************************************************************/
//                              State Variables                      //
/*********************************************************************/
//float   util[32]:     Utilization of the link.
//int     link_id[32]:  Objid's of all the links attached to the switch.
//int     total_bits[32]: Total number of bits to pass through the assoc link.
```

```
/*****************************************************************/
//                        Temporary Variables                    //
/*****************************************************************/
//                              NONE

//end set_link_cost
//*****************************************************************//
```

10. TYPE STATE

The following code is contained in the **type** state in the Fibre Channel switch FSM.

```
//*****************************************************************//
//Type state determines if the packet is a primative or a frame

streamIn = op_intrpt_strm();  //which stream did it come from?
pkptr = op_pk_get (streamIn);

//if numFields < 3 then its an r_rdy
//else if numFields > 3 its a frame
op_pk_format_info_get(pkptr,OPC_PK_PROPERTY_NUMBER_FIELDS,
                              &numFields);
/*****************************************************************/
//                        Global Variables                       //
/*****************************************************************/
//                              NONE
/*****************************************************************/
//                        State Variables                        //
/*****************************************************************/
//                              NONE
```

```
/*****************************************************************/
//                        Temporary Variables                    //
/*****************************************************************/

//Packet*       pkptr:   The pointer to the active packet.
//int           numFields:     The number of fields in the frame.
//int           streamIn:      The stream that the packet came in on


//end type
//*****************************************************************//
```

11. R_RDY STATE

The following code is contained in the **r_rdy** state in the Fibre Channel switch FSM.

```
//*****************************************************************//
//If you got an rrdy increment the b2b credit
//get the src node from the r_rdy and put it into src_nde

op_pk_nfd_get (pkptr, "src_nde", &src_nde); // where did it come from?

inc_buf = get_outStream(node_id, link_id, src_nde);  //increment the buffer assoc
                                                     //with the incomming rrdy
if (buf_to_buf[inc_buf] < max_credit)        {
        buf_to_buf[inc_buf] == buf_to_buf[inc_buf]++;
}       else {
                buf_to_buf[inc_buf] = max_credit;  //can never be > than max_credit
}
op_pk_destroy(pkptr);
/*****************************************************************/
//                        Global Variables                       //
/*****************************************************************/
//                              NONE
```

```
/*****************************************************************/
//                        State Variables                        //
/*****************************************************************/

//int    buf_to_buf[32]: Holds the buffer to buffer credit of each attached node.
//int    node_id[32]:  Objid's of all the nodes attached to the switch.
//int    link_id[32]:  Objid's of all the links attached to the switch.
//int    max_credit:          The login buf_to_buf credit.
/*****************************************************************/
//                      Temporary Variables                      //
/*****************************************************************/

//Packet*      pkptr:    The pointer to the active packet.
//int*         src_nde:  The node that just sent the packet.
//                       This is where the r_rdy is returned to.
//int          inc_buf:  Inc the buf of the node that sent the r_rdy.


//end r_rdy
//*****************************************************************//
```

12. FRAME STATE

The following code is contained in the **frame** state in the Fibre Channel switch FSM.

```
//*****************************************************************//
//Frame advances the route pointer and helps with debug

op_rte_pk_advance(pkptr);

op_rte_pk_src_node (pkptr, &subnet_id_ptr, &node_id_ptr);
printf("My source is (%d, %d) \n", subnet_id_ptr, node_id_ptr);

op_rte_pk_dst_node (pkptr, &subnet_id_ptr, &node_id_ptr);
printf("My destination is (%d, %d) \n", subnet_id_ptr, node_id_ptr);
```

```
op_rte_pk_next_node (pkptr, &next_subnet, &next_node);
printf("I am the switch and my number is %d \n",s_id);
printf("The next node is:  %d \n", next_node);
/*****************************************************************/
//                          Global Variables                     //
/*****************************************************************/
//                              NONE
/*****************************************************************/
//                          State Variables                      //
/*****************************************************************/
//                              NONE
/*****************************************************************/
//                          Temporary Variables                  //
/*****************************************************************/
//Packet*       pkptr:   The pointer to the active packet.
//int*          subnet_id_ptr: Used to temp hold subnet id's
//int*          node_id_ptr:   Used to temp hold node id's
//int*          next_subnet:   Used to temp hold the next subnet
//int*          next_node:     Used to temp hold the next node id

//end frame
//*****************************************************************//
```

13. DEC_BUF STATE

The following code is contained in the **dec_buf** state in the Fibre Channel switch FSM.

```
//*****************************************************************//
//Dec_buf state checks to ensure that there is credit with the next node on the route.
//If there is a busy fabric, statistics are recorded.

//Ensure the correct transition to the next state
credit = -1;
class3 = -1;

// where is it going to?
dec_buf = get_outStream(node_id, link_id, next_node);

if (buf_to_buf[dec_buf] >= 1){  //   if you have credit with the receiving node
        buf_to_buf[dec_buf] == buf_to_buf[dec_buf]--;
        //decrement the credit in that buffer
        credit = 1;                                     //transition to snd_rrdy
}       else    {                       //no credit with the next node indicates a busy fabric
        op_pk_nfd_get(pkptr,"SOF_3",&classType); //what class is the frame,
                                                  //no f_bsy to class3!
                if (classType == 0x56)          {       //if it's a class 3 destroy the packet
                                                //and wait for the next frame to arrive.
                op_pk_destroy(pkptr);           //destroy the packet
                DED_CL_3 = DED_CL_3 ++;         //increment the number of class 3's
                                                //destroyed by this switch
                op_stat_write (ded_cl_3_gsh, DED_CL_3); //write the stat
                class3 = 1;                             //return to idle

        //      I have no credit and I'm a class 1/2 frame
        }       else if (classType == 0x57 || classType == 0x55)  {
                        credit = 0;                                     //go to f_bsy
        }
}
```

```
/*****************************************************************/
//                        Global Variables                       //
/*****************************************************************/

//int          DED_CL_3:  Stat for number of class 3 frames destroyed by the switch.
/*****************************************************************/
//                        State Variables                        //
/*****************************************************************/

//Stathandle  ded_cl_3_gsh:  SH which holds the value of DED_CL_3
//int         node_id[32]:   Objid's of all the nodes atch to the sw.

//int         link_id[32]:   Objid's of all the links atch to the sw.
/*****************************************************************/
//                      Temporary Variables                      //
/*****************************************************************/

//Packet*  pkptr:     The pointer to the active packet.
//int*     next_node: Used to temp hold the next node id
//int      credit:    Used to help the transition to f_bsy or sndrrdy.

//int      class3:    Used to transition to idle if no credit and a class three Frame.

//int      classType: Holds the class of the Frame.

//int      dec_buf:   Index to the buf_to_buf array. Tells which buffer to reduce by one.


//end dec_buf
//*****************************************************************//
```

14. F_BSY STATE

The following code is contained in the **f_bsy** state in the Fibre Channel switch FSM.

```
//*****************************************************************//
//f_bsy determines if it is an F_BSY to data or to a LCF

//find out if it's a FC-0 (LCF) or FC-1 (DATA)
op_pk_nfd_get(pkptr,"R_Bits", &fc_type);
```

```
//swap the D_ID/S_ID  so you can return it
op_pk_nfd_get(pkptr, "S_ID", &temp_sid);
op_pk_nfd_get(pkptr, "D_ID", &temp_did);
op_pk_nfd_set(pkptr, "S_ID", temp_did);
op_pk_nfd_set(pkptr, "D_ID", temp_sid);

if (fc_type >= 0x2 && fc_type < 0xC)        { //if it's a data frame
        op_pk_nfd_set(pkptr, "R_Bits", 0xC); //make it an Link Control Frame
        op_pk_nfd_set(pkptr, "INFO", 0x5);  //make it an F_BSY to data
        op_pk_bulk_size_set(pkptr,0);   //make the data size zero
}       else if (fc_type == 0xC)        {       //if it's a link control frame
        op_pk_nfd_set(pkptr, "INFO", 0x6);      //make it an F_BSY to LCF
        op_pk_nfd_set(pkptr, "RX_ID", 0x00);        //give it an exchange ID
        }
/*************************************************************************/
//                              Global Variables                         //
/*************************************************************************/
//                                  NONE
/*************************************************************************/
//                              State Variables                          //
/*************************************************************************/
//                                  NONE
/*************************************************************************/
//                              Temporary Variables                      //
/*************************************************************************/
//Packet*       pkptr:    The pointer to the active packet.
//int           fc_type:  Data type or Link Control Frame.
//int           temp_sid: Temp holder for the s_id.
//int           temp_did: Temp holder for the d_id.

//end f_bsy
//*************************************************************************//
```

15. ADD_ROUTE STATE

The following code is contained in the **add_route** state in the Fibre Channel switch FSM.

```
//*********************************************************************//
//add_route adds a return route to the F_BSY generated by the switch

op_pk_nfd_get(pkptr, "D_ID", &d_id);        //get the  destination
dst_subnet = op_topo_parent(d_id);          //get the subnet of that destination

rset_ptr = op_rte_routeset_create (TOPO_PTR, subnet, s_id,
                                   dst_subnet, d_id, MAX_HOPS);
tmp_rte = op_rte_routeset_mincost (rset_ptr);
rte_arry =  op_rte_route_copy (tmp_rte);                    //copy to a temporary route
op_rte_pk_route_insert(pkptr, tmp_rte);          //add the route to the packet
op_rte_pk_next_node (pkptr, &next_subnet, &next_node);  //find out the next node
/*********************************************************************/
//                          Global Variables                         //
/*********************************************************************/
//extern      Topology*      TOPO_P:  The one and only Topology pointer.
//extern      const int      MAX_HOPS:  Max number of hops a packet can take.
/*********************************************************************/
//                          State Variables                          //
/*********************************************************************/
//Route_Set*  rset_ptr:  The route set pointer for returning packets.
//Route*      rte_arry:  The route for the fbsy from the switch to its destination.
//int         s_id:      The source id of this node.
//int         subnet:    The subnet of the assoc node.


/*********************************************************************/
```

```
//                          Temporary Variables                          //
/*****************************************************************/

//Packet*       pkptr:   The pointer to the active packet.
//Route*        tmp_rte:    The route which is inserted on the packet.
//int*          next_node:   Used to temp hold the next node id.
//int           next_subnet: The subnet of the next node.
//int           d_id:           The destination.
//int           dst_subnet:  The subnet of the destination.

//end add_route
//*****************************************************************//
```

16. SNDRRDY STATE

The following code is contained in the **sndrrdy** state in the Fibre Channel switch FSM.

```
//*****************************************************************//
//      snd rrdy builds an rrdy and sends it back to the previous node that sent the frame.

rdyptr = op_pk_create_fmt ("R_RDY");

//get the src node from the frame and put it into src_nde
op_pk_nfd_get (pkptr, "src id", &src_nde);
printf("the previous node is %d\n",src_nde);
printf("MY NODE IS %d \n",s_id);

//put your s_id in the "src nde" field
op_pk_nfd_set (rdyptr, "src_nde", s_id);

//      Find the packet stream connected to src_nde
streamOut = get_outStream(node_id, link_id, src_nde);
printf("rrdy streamOUT = %d\n", streamOut);
printf("i'm going to = %d, nde_ary = %d\n", src_nde, node_id[streamOut]);
```

```
//the following data will be used to calculate utiliztion
total_bits[streamOut] = total_bits[streamOut] + op_pk_total_size_get (rdyptr);


//      set some delay based on the time it took to get to this point
// i.e. had to process all the stuff up to now. which took time.
// if one wanted to have some PDF based delay this would be
// where that delay would be placed.
// for example op_pk_send_delayed(rdyptr,streamOut,
//                                      op_dist_exponential(r_rdy_delay));

if (r_rdy_delay != 0 ) {                //if there is a non-zero delay
//send it out the transmitter with the proscribed delay
        op_pk_send_delayed (rdyptr, streamOut, r_rdy_delay);
} else {
//send it with no delay
        op_pk_send (rdyptr, streamOut);
}
/**************************************************************************/
//                              Global Variables                          //
/**************************************************************************/
//                              NONE
/**************************************************************************/
//                              State Variables                           //
/**************************************************************************/
//double  r_rdy_delay:  Delay required to process, build and send an R_RDY.
//double  total_bits[32]: Total number of bits to pass through the assoc link.
//int     node_id[32]:  Objid's of all the nodes attached to the switch.
//int     link_id[32]:  Objid's of all the links attached to the switch.
/**************************************************************************/
//                              Temporary Variables                       //
/**************************************************************************/
//Packet*       pkptr:  The pointer to the active packet.
```

```
//Packet*       rdyptr:      The pointer used to create the R_RDY primative.
//int*          src_nde:     The node that just sent the packet.
//                           This is where the r_rdy is returned to.
//int           streamOut:   Index of the stream where the R_RDY is going.

//end sndrrdy
//*************************************************************************//
```

17. XMT

The following code is contained in the **xmt** state in the Fibre Channel switch FSM.

```
//*************************************************************************//
//xmt is used to send the frame to its next destination

op_pk_nfd_set (pkptr, "src id", s_id);  //put your sid on the frame

//find the stream that has the next node attached to it
streamOut = get_outStream(node_id, link_id, next_node);
printf("The output stream index is %d\n", streamOut);
printf("next_node = %d, nde_ary = %d\n", next_node, node_id[streamOut]);

//the following data will be used to calculate utiliztion
total_bits[streamOut] = total_bits[streamOut] + op_pk_total_size_get (pkptr);

if (proc_delay != 0 )   {    //if the delay is non-zero
//send it out the transmitter with the proscribed delay
//this would be a good place to input a PDF for proc_delay
//for now we will use a const proc_delay.
```

```
        op_pk_send_delayed (pkptr, streamOut, proc_delay);
        } else {                    //send it with no delay
        op_pk_send (pkptr, streamOut);
}

/**************************************************************************/
//                              Global Variables                          //
/**************************************************************************/
//                                  NONE
/**************************************************************************/
//                              State Variables                           //
/**************************************************************************/
//double  proc_delay:  Delay required to process, build and send a frame.
//int     node_id[32]: Objid's of all the nodes attached to the switch.
//int     link_id[32]:  Objid's of all the links attached to the switch.
//int     total_bits[32]: Total number of bits to pass through the assoc link.
/**************************************************************************/
//                              Temporary Variables                       //
/**************************************************************************/
//Packet*       pkptr:  The pointer to the active packet.
//int*          next_node:    Used to temp hold the next node id
//int           streamOut:    Index of the stream where the R_RDY is going.

//end xmt
//************************************************************************//
```

APPENDIX D. MATLAB CODE USED DURING SIMULATION

1. LEAST SQUARES FIT OF SETTLING TIME DATA

```
function coef = settletime (DEL_COST, Settle_time)

order = 1;              %order of the polynomial

% Now use LSQ fit to get slope and y intercept

eq_of_line = polyfit(DEL_COST, Settle_time, order);


t = (0:.01:2);    %used to show curve fit for t = 0:2

y = eq_of_line(1)*t + eq_of_line(2);  % y = mx + b

%Theory shows what the polyfit would get w/
%other values of DEL_COST

theory = eq_of_line(1)*DEL_COST + eq_of_line(2);

% Table to show the data
table =[DEL_COST Settle_time theory . . .
        (Settle_time - theory)]

%diff = absolute difference btwn theory and simulation
diff = abs (table(:,4))

%average difference btwn theory and simulation
mean_diff = mean(diff)

coef = eq_of_line;
m=num2str(coef(1)); %Slope of the line
b=num2str(coef(2)); %Y intercept of the line

figure(1)

%plotted using log-log for clarity
loglog (t, y, DEL_COST, Settle_time, 'ro')
xlabel('DEL\_COST (sec) = x')
ylabel('Settling Time (sec) = y')
legend(['y = ',m,'x ',b],'Simulation Data',2)
grid
```

2. CALCULATION OF TIME TO R_RDY

```
function time = time2rrdy (frameSize, dist, refIndex,
                           rrdyDly)


%  time = time2rrdy (frameSize, dist, refIndex, rrdyDly)
%
%  frameSize is the size in Bits of the Data Payload
%  dist is the one way distance from the node to the switch
%  refIndex is the refraction index of the optical fiber
%  rrdyDly is the time required for the switch to respond to
%the frame with an R_RDY.
%
%  This function calculates the time required for a node
%to receive an R_RDY reply from a switch after sending a
%frame.

%first define the constants
tt1b = 1/1062500000;    %time to transmit one bit
propSpeed = 3.0e8./refIndex  % speed of light in the medium

rrdyTime = tt1b*40;     %time to transmit the R_RDY
frameXmtTime = frameSize*tt1b; %xmission delay of the frame
propTime = dist./propSpeed;  %Prop delay through the fiber

%Total time calculation
time = rrdyTime + 2*propTime + frameXmtTime +rrdyDly;
```

3. CALCULATION OF REQUIRED CREDIT

```
function credit = calcCredit(secPerFrame, returnDelay)

%credit = calcCredit(interarrival_arg, returnDelay)
%
%For this function you should enter the interarrrival
%argument for the secPerFrame and either the time required
%for the node of interest to receive a R_RDY from the
%switch for a Buffer-to-Buffer calculation.
%Or, for an End-to-End credit calculation use the end to
%end delay through the system of interest.


credit = ceil(returnDelay./secPerFrame);
```

4. INTERARRIVAL ATTRIBUTE CALCULATION

```
function iarr = iarrCalc(throughput,dataBitsPerFrame)

%iarr = iarrCalc(throughput, dataBitsPerFrame)
%
%  throughput is the desired output in BPS of the
%node of interest
%  dataBitsPerFrame is the number of payload bits
%that are expected per frame.
%
%This function calculates the interarrival argument
%for the ideal packet generator in the node model.
%This function allows the designer to take a data
%throughput requirement and translate it into an
%attribute in the simulation node model.

%This does the 8b/10b encoding plus header
bitsPerFrame = dataBitsPerFrame*10/8 + 600;

%This calculates the actual Fibre Channel
%througput including the overhead of the
%header and that caused by 8b/10b encoding.
reqtput = throughput.*bitsPerFrame./dataBitsPerFrame;

disp('Required Fibre Channel Througput is: ')
disp(reqtput)

disp('This represents a utilization of: ')
disp(reqtput/1062500000)

%Frame interarrival rate is one over the reqired
%throughput.
iarr = (1./reqtput).*bitsPerFrame;
```

5. OPTIMAL CREDIT CALCULATION FUNCTION

```
function credit = calcCredit(secPerFrame, returnDelay)

%credit = calcCredit(interarrival_arg, returnDelay)
%
%For this function you should enter the interarrrival
%argument for the secPerFrame and either the time
%required for the node of interest to receive a R_RDY
%from the switch for a Buffer-to-Buffer calculation.
%Or, for an End-to-End credit calculation use the end
%to end delay through the system of interest.


credit = ceil(returnDelay./secPerFrame);
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

1. Joint Advanced Strike Technology Program, *Avionics Architecture Definition, Version 1.0*, 1994.

2. Anthony Anderberg, "History of the Internet and Web," http://www.geocities.com/~anderberg/ant/history/, November 1999.

3. SBS Aviation Technologies, "An Interpretation of MIL-STD-1553B," ftp://ftp.sbse.com/web_files/pdf/general/1553Interpretation.pdf, November 1999.

4. Robert W. Kembel, *Comprehensive Introduction to Fibre Channel,* Northwest Learning Associates, Tucson, Arizona 1998.

5. Valerie Illingsworth, Dictionary of Computing, Oxford University Press, London, January 1997.

6. Phil Dowe, "CCC Class Notes," http://info.utas.edu.au/humsoc/philosophy/Phil_Dowe/, University of Tasmania, Australia 1999.

7. NAVSEA Acquisition Support Office, "Acquisition Information Memorandum: Special Edition Open Systems Approach," http://www.acq-ref.navy.mil/opensyst.html, January 1997.

8. Ed Lee, "Introduction to Fibre Channel," Fibre Channel Group, November 1999.

9. BYTE Magazine, "Five Problems Fibre Channel Solves," http://www.byte.com/art/9605/img/056netb2.htm, May 1999.

10. Zoltan Meggyesi, "Fibre Channel Overview," http://www1.cern.ch/HIS/fcs/spec/overview.htm, November 1999.

11. American National Standards Institute, "About ANSI," http://web.ansi.org/public/about.html, November 1999.

12. Technical Committee T11, http://www.t11.org/, November 1999.

13. Mike Dorsett, "Fibre Channel Avionics Environment Profile," ftp://ftp.t11.org/t11/pub/fc/ae/98-303v0.htm, June 1998.

14. Wilf Sullivan, "Fibre Channel Replacement for MIL-STD-1553 and Next Generation Military Data Bus," DY Systems Inc., Ontario, Canada, December 1997.

15. The Fibre Channel Industry Association, "Fibre Channel Overview: Technology Comparison," http://www.fibrechannel.com/technology/compare.htm, November 1999.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Rd., Suite 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, CA 93943-5101

3. OPNET Technologies 1
   Attn: Alain Cohen, CTO
   3400 International Drive NW
   Washington, DC 20008

4. Joint Strike Fighter Program Office 2
   Attn: CDR Scott Orren
   47123 Buse Rd., Bldg. 2272, Rm. 146
   Alexandria, VA 20670

5. Prof. Max F. Platzer, Code AA/Pl 1
   Chairman, Department of Aeronautics and Astronautics
   Naval Postgraduate School
   Monterey, CA 93943-5000

6. Prof. Russell W. Duren, Code AA/Dr 3
   Department of Aeronautics and Astronautics
   Naval Postgraduate School
   Monterey, CA 93943-5000

7. Prof. John C. McEachen, Code EC/Mj 1
   Department of Aeronautics and Astronautics
   Naval Postgraduate School
   Monterey, CA 93943-5000

8. Boeing Aerospace Co 1
   Attn: Mike Foster, Mail Stop 85-18
   P.O. Box 3999
   Seattle, WA 98124

9. Naval Information Warfare Analysis Center 1
Attn: Rosemary Wenchel (NIWA)
9800 Savage Road
Ft Mead, MD 20755

10. LCDR Shawn P. Hendricks, USN 1
8698 S. Aberdeen Circle
Highlands Ranch, CO 80130